优炫数据库特性参考手册 2.1

优炫数据库特性参考手册 2.1

版权 © 2016-2023 北京优炫软件股份有限公司

法律声明

优炫数据库管理系统(简称：UXDB) 是由北京优炫软件股份有限公司开发并发布的一款商业性数据库管理系统。

优炫数据库管理系统(UXDB)的一切知识产权以及与该软件产品相关的所有信息内容，包括但不限：文字表述及其组合、图标、图饰、图表、色彩、界面设计、版面框架、有关数据、及电子文档等均属北京优炫软件股份有限公司所有。本软件及其文档的任何使用、复制、修改、出租、传播、销售及分发等行为均须经北京优炫软件股份有限公司书面许可。

凡侵犯北京优炫软件股份有限公司知识产权的行为，北京优炫软件股份有限公司将依法追究其法律责任。

本声明的最终解释权归属于北京优炫软件股份有限公司。

和其他优炫公司商标均为北京优炫软件股份有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

由于产品版本安装或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

北京优炫软件股份有限公司（总部）

- 地址：北京市海淀区学院南路62号中关村资本大厦11层（邮编：100081）
- 网址：http://www.uxsino.com
- 邮箱：<uxdb_support@uxsino.com>
- 电话：010-82886998
- 传真：010-82886338
- 服务热线：400-650-7837

目录

前言 xi
1. 文档目的 xi
2. 文档对象 xi
3. 修改记录 xi
1. 概述 1
2. 中文全文检索 2
2.1. 概述 2
2.1.1. SCWS 2
2.1.2. zhparser 4
2.2. 安装与配置 5
2.2.1. 安装zhparser 5
2.2.2. 获得解析器定义的token类型 6
2.2.3. 配置实例 7
2.3. 示例 8
2.3.1. 解析器示例 8
2.3.2. tsvector转化示例 9
2.3.3. tsquery转化示例 9
2.3.4. 查询示例 9
2.3.5. 中文全文检索示例 9
2.4. 问题与解决 11
3. 闪回删除 12
3.1. 概述 12
3.2. 原理 12
3.3. 示例 13
4. 闪回表 15
4.1. 概述 15
4.2. 原理 15
4.3. 示例 16
5. btree_gin 19
5.1. 概述 19
5.2. 示例 19
6. btree_gist 20
6.1. 概述 20
6.2. 示例 20
7. cstore_fdw 22
7.1. 概述 22
7.2. 示例 22
8. dblink 25
8.1. dblink_connect 25
8.1.1. 大纲 25
8.1.2. 描述 26
8.1.3. 参数 26
8.1.4. 返回值 26
8.1.5. 注解 26
8.1.6. 示例 26
8.2. dblink_connect_u 27
8.2.1. 大纲 27
8.2.2. 描述 27
8.3. dblink_disconnect 28
8.3.1. 大纲 28
8.3.2. 描述 28

8.3.3. 参数 28
8.3.4. 返回值 28
8.3.5. 示例 28
8.4. dblink 28
8.4.1. 大纲 29
8.4.2. 描述 29
8.4.3. 参数 29
8.4.4. 返回值 29
8.4.5. 注解 30
8.4.6. 示例 30
8.5. dblink_exec 31
8.5.1. 大纲 31
8.5.2. 描述 31
8.5.3. 参数 31
8.5.4. 返回值 32
8.5.5. 示例 32
8.6. dblink_open 32
8.6.1. 大纲 32
8.6.2. 描述 33
8.6.3. 参数 33
8.6.4. 返回值 33
8.6.5. 注解 33
8.6.6. 示例 33
8.7. dblink_fetch 33
8.7.1. 大纲 34
8.7.2. 描述 34
8.7.3. 参数 34
8.7.4. 返回值 34
8.7.5. 注解 34
8.7.6. 示例 34
8.8. dblink_close 35
8.8.1. 大纲 35
8.8.2. 描述 35
8.8.3. 参数 35
8.8.4. 返回值 36
8.8.5. 注解 36
8.8.6. 示例 36
8.9. dblink_get_connections 36
8.9.1. 大纲 36
8.9.2. 描述 36
8.9.3. 返回值 36
8.9.4. 示例 36
8.10. dblink_error_message 37
8.10.1. 大纲 37
8.10.2. 描述 37
8.10.3. 参数 37
8.10.4. 返回值 37
8.10.5. 示例 37
8.11. dblink_send_query 37
8.11.1. 大纲 38
8.11.2. 描述 38
8.11.3. 参数 38
8.11.4. 返回值 38
8.11.5. 示例 38

8.12. dblink_is_busy 38
8.12.1. 大纲 38
8.12.2. 描述 38
8.12.3. 参数 38
8.12.4. 返回值 39
8.12.5. 示例 39
8.13. dblink_get_notify 39
8.13.1. 大纲 39
8.13.2. 描述 39
8.13.3. 参数 39
8.13.4. 返回值 39
8.13.5. 示例 39
8.14. dblink_get_result 40
8.14.1. 大纲 40
8.14.2. 描述 40
8.14.3. 参数 40
8.14.4. 返回值 40
8.14.5. 注解 40
8.14.6. 示例 40
8.15. dblink_cancel_query 41
8.15.1. 大纲 41
8.15.2. 描述 42
8.15.3. 参数 42
8.15.4. 返回值 42
8.15.5. 示例 42
8.16. dblink_get_pkey 42
8.16.1. 大纲 42
8.16.2. 描述 42
8.16.3. 参数 42
8.16.4. 返回值 43
8.16.5. 示例 43
8.17. dblink_build_sql_insert 43
8.17.1. 大纲 43
8.17.2. 描述 43
8.17.3. 参数 43
8.17.4. 返回值 44
8.17.5. 示例 44
8.18. dblink_build_sql_delete 44
8.18.1. 大纲 44
8.18.2. 描述 44
8.18.3. 参数 44
8.18.4. 返回值 45
8.18.5. 示例 45
8.19. dblink_build_sql_update 45
8.19.1. 大纲 45
8.19.2. 描述 45
8.19.3. 参数 45
8.19.4. 返回值 46
8.19.5. 示例 46
9. mysql_fdw 47
9.1. 概述 47
9.2. 依赖工具 47
9.3. 示例 47
10. oracle_fdw 49

10.1. 概述 49
10.2. 依赖工具 49
10.3. 选项 49
10.3.1. 外部数据包装器选项 49
10.3.2. 外部服务器选项 49
10.3.3. 用户映射选项 50
10.3.4. 外部表选项 50
10.3.5. 列选项 51
10.4. 示例 51
11. orafce 54
11.1. 概述 54
11.2. 示例 54
11.3. 支持特性 54
11.3.1. 数据类型 54
11.3.2. 系统视图和dual表 55
11.3.3. 函数 56
11.3.4. 操作符 59
11.3.5. 支持的包 60
11.3.6. 兼容Oracle数据库功能 68
11.4. 扩展特性 70
11.4.1. 函数兼容 70
12. ux_implicit 77
12.1. 概述 77
12.2. 隐式转换类型 77
12.3. 支持场景 78
12.4. 示例 86
13. postgres_adaptor 88
14. tablefunc 90
14.1. normal_rand 90
14.2. crosstab(text) 91
14.3. crosstabN(text) 93
14.4. crosstab(text, text) 94
14.5. connectby 96
15. tds_fdw 99
15.1. 概述 99
15.2. 依赖工具 99
15.3. 示例 99
16. uxAgent 101
16.1. 安装依赖软件 101
16.2. uxAgent的安装 101
16.2.1. Linux下uxAgent的安装 101
16.2.2. Windows下uxAgent的安装 103
16.3. 示例 107
16.3.1. 创建定时任务 107
16.3.2. 元数据表 109
17. uxaudit 112
17.1. 审计表 112
17.2. 钩子函数 114
17.3. 示例 115
18. uxcrypto 116
18.1. 普通哈希函数 116
18.1.1. digest() 116
18.1.2. hmac() 116
18.2. 口令哈希函数 116

18.2.1. crypt() 117
18.2.2. gen_salt() 117
18.3. PGP加密函数 118
18.3.1. pgp_sym_encrypt() 119
18.3.2. pgp_sym_decrypt() 119
18.3.3. pgp_pub_encrypt() 119
18.3.4. pgp_pub_decrypt() 119
18.3.5. pgp_key_id() 120
18.3.6. armor(), dearmor() 120
18.3.7. pgp_armor_headers 120
18.3.8. PGP函数的选项 120
18.3.9. 用GnuPG生成PGP密钥 122
18.3.10. PGP代码的限制 123
18.4. 原始的加密函数 123
18.5. 随机数据函数 124
18.6. 注解 124
18.6.1. NULL处理 124
18.6.2. 安全性限制 124
19. uxdb_fdw 125
19.1. fdw选项 125
19.1.1. 连接选项 125
19.1.2. 对象名称选项 126
19.1.3. 代价估计选项 126
19.1.4. 远程执行选项 127
19.1.5. 可更新性选项 127
19.1.6. 导入选项 127
19.2. 连接管理 128
19.3. 事务管理 128
19.4. 远程查询优化 128
19.5. 远程查询执行环境 129
19.6. 示例 129
20. uxgis 131
20.1. 依赖库 131
20.2. 示例 131
21. uxrowlocks 137
21.1. 概述 137
21.2. 示例 138
22. uxstattuple 139
22.1. uxstattuple(regclass) returns record 139
22.2. uxstatindex(regclass) returns record 140
22.3. uxstatginindex(regclass) returns record 140
22.4. uxstathashindex(regclass) returns record 141
22.5. ux_relpages(regclass) returns bigint 141
22.6. uxstattuple_approx(regclass) returns record 142
23. ux_cron 144
23.1. 配置 144
23.2. 示例 144
24. ux_hint_plan 147
24.1. 概述 147
24.2. 语法 147
24.3. 示例 148
25. ux_prewarm 151
26. ux_stat_statements 152
26.1. ux_stat_statements视图 152

26.2. 函数 154
26.2.1. ux_stat_statements_reset() returns void 154
26.2.2. ux_stat_statements(showtext boolean) returns setof record . 154
26.3. 配置参数 154
26.3.1. ux_stat_statements.max (integer) 154
26.3.2. ux_stat_statements.track (enum) 155
26.3.3. ux_stat_statements.track_utility (boolean) 155
26.3.4. ux_stat_statements.save (boolean) 155
26.4. 示例 155
27. xml2 157
27.1. 函数 157
27.2. xpath_table 158
27.2.1. 多值结果 159
27.3. XSLT函数 160

表格清单

1. 文档更新记录 ... xi
2.1. 文本词典格式 ... 4
2.2. token类型 ... 6
8.1. dblink功能 ... 25
8.2. dblink_connect 参数说明 ... 26
8.3. dblink_disconnect 参数说明 ... 28
8.4. dblink 参数说明 ... 29
8.5. dblink_exec 参数说明 ... 31
8.6. dblink_open 参数说明 ... 33
8.7. dblink_fetch 参数说明 ... 34
8.8. dblink_close 参数说明 ... 35
8.9. dblink_error_message 参数说明 ... 37
8.10. dblink_send_query 参数说明 ... 38
8.11. dblink_is_busy 参数说明 ... 38
8.12. dblink_get_notify 参数说明 ... 39
8.13. dblink_get_result 参数说明 ... 40
8.14. dblink_cancel_query 参数说明 ... 42
8.15. dblink_get_pkey 参数说明 ... 42
8.16. dblink_build_sql_insert 参数说明 ... 43
8.17. dblink_build_sql_delete 参数说明 ... 44
8.18. dblink_build_sql_update 参数说明 ... 45
11.1. 数据类型 ... 55
11.2. 系统视图和dual表 ... 55
11.3. dbms_pipe功能表 ... 61
11.4. dbms_alert功能表 ... 62
11.5. PLVdate功能表 ... 63
11.6. PLVstr and PLVchr功能表 ... 64
11.7. DBMS_utility功能表 ... 66
11.8. DBMS_ASSERT功能表 ... 67
11.9. PLUnit功能表 ... 67
11.10. DBMS_random功能表 ... 67
11.11. dateadd 参数说明 ... 70
11.12. datediff 参数说明 ... 71
11.13. to_char函数转换模板 ... 74
12.1. 字符类型到数字类型 ... 77
12.2. 字符类型到时间类型和时间间隔类型 ... 77
12.3. 布尔类型到数字类型 ... 77
12.4. 数字类型到字符类型 ... 77
12.5. 时间类型和时间间隔类型到字符类型 ... 77
12.6. 字符类型到布尔类型 ... 77
12.7. 隐式转换的函数及方向 ... 79
12.8. 隐式转换的操作符及方向 ... 84
13.1. 输入适配与输出适配 ... 88
14.1. tablefunc函数 ... 90
14.2. connectby参数 ... 96
17.1. log_event ... 112
17.2. session ... 113
17.3. logon ... 113
17.4. audit_statement ... 113
17.5. audit_substatement ... 114
17.6. audit_substatement_detail ... 114

17.7. 钩子函数 114
18.1. crypt()支持的算法 117
18.2. crypt()的迭代计数 117
18.3. 哈希算法速度 118
18.4. type字符串参数说明 123
21.1. uxrowlocks输出列 137
22.1. uxstattuple输出列 139
22.2. uxstatindex输出列 140
22.3. uxstatginindex输出列 141
22.4. uxstathashindex输出列 141
22.5. uxstattuple_approx输出列 142
24.1. hint提示语句 147
26.1. ux_stat_statements列 152
27.1. 函数 157
27.2. xpath_table参数 158

前言

1. 文档目的

本文档介绍UXDB数据库特性，为相关技术人员和用户提供了必要的参考。

2. 文档对象

- 技术支持工程师
- 维护工程师
- 优炫数据库用户

3. 修改记录

修改记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

表 1. 文档更新记录

工具版本	发布日期	修改说明
2.1.1.5C	2022-12-08	第一次正式发布。

第 1 章 概述

本文档介绍UXDB安装目录下uxdbinstall/dbsql/share/extension中的服务器插件模块。

许多模块提供新的用户自定义函数、操作符或数据类型。为了使用这些模块，需要执行CREATE EXTENSION命令：

CREATE EXTENSION module_name;

这个命令必须由数据库系统管理员运行。CREATE EXTENSION只会把新的SQL对象注册在当前数据库中，因此需要在每一个希望使用该模块功能的数据库中执行这个命令。另外，可以在template1数据库中运行这个命令以便该扩展能被默认地复制到后续创建的数据库中。

很多模块允许将它们的对象安装在选择的模式中。要这样做，需要将SCHEMA schema_name加入到CREATE EXTENSION命令中。默认情况下，这些对象将被放置在当前创建目标模式中，通常是public。

第 2 章 中文全文检索

2.1. 概述

UXDB支持全文检索，其内置的缺省的分词解析器采用空格分词。因为一般英语等语言分词比较简单，按照标点、空格切分语句即可获得有含义的词语，UXDB自带的解析器就是按照这个原理来分词的，比较简单。而中文比较复杂，词语之间没有空格分割，长度也不固定，分词有时还跟语句的语义相关，因此自带的解析器不能进行中文分词。

要支持中文的全文检索需要额外的中文分词插件，zhparser是基于简易中文分词系统(SCWS-Simple Chinese Word Segmentation)实现的一个插件。zhparser用C语言实现了UXDB TEXT SEARCH PARSER需要的接口，这些接口会调用SCWS中文分词引擎进行分词。使用zhparser这个插件，便可以使UXDB支持中文分词，继而可以使用UXDB支持中文全文检索。

2.1.1. SCWS

SCWS是简易中文分词系统，全称为Simple Chinese Word Segmentation。

这是一套基于词频词典的机械式中文分词引擎，它能将一整段的中文文本基本正确地切分成词。词是中文的最小语素单位，但在书写时并不像英语会在词之间用空格分开，所以如何准确并快速分词一直是中文分词的攻关难点。

SCWS采用纯C语言开发，不依赖任何外部库函数，可直接使用动态链接库嵌入应用程序，支持的中文编码包括GBK、UTF-8。此外还提供了PHP扩展模块，可在PHP中快速而方便地使用分词功能。

分词算法上并无太多创新成分，采用的是自己采集的词频词典，并辅以一定的专有名称，人名，地名，数字年代等规则识别来达到基本分词，经小范围测试准确率在90% ~ 95%之间，基本上能满足一些小型搜索引擎、关键字提取等场合运用。

UXDB默认安装SCWS（安装目录：uxdbinstall/thirdparty/scws/bin）。

2.1.1.1. scws分词命令行工具

可以在命令行中使用scws命令进行测试分词。执行scws -h可以看到详细帮助说明。

scws ［*option*］［[-i］*input*］［[-o］*output*］

-i *string|file*

要切分的字符串或文件，如不指定则程序自动读取标准输入，每输入一行执行一次分词。

-o *file*

切分结果输出保存的文件路径，若不指定直接输出到屏幕。

-c *charset*

指定分词的字符集，默认是gbk，可选utf8。

-r *file*

指定规则集文件（规则集用于数词、数字、专有名称、人名的识别）。

-d *file[:file2[:...]]*

指定词典文件路径（XDB格式，请在-c之后使用）。

自1.1.0起，支持多词典同时载入，也支持纯文本词典（必须是.txt结尾），多词典路径之间用冒号隔开，排在越后面的词典优先级越高。

文本词典的数据格式参见scws-gen-dict所用的格式，但更宽松一些，允许用不定量的空格分开，只有<词>是必备项目，其它数据可有可无，当词性标注为“!”（叹号）时表示该词作废，即使在较低优先级的词库中存在该词也将作废。

-M *level*

复合分词的级别：1~15，按位异或的1|2|4|8　依次表示短词|二元|主要字|全部字，缺省不复合分词。

-I

输出结果忽略跳过所有的标点符号。

-A

显示词性。

-E

将xdb词典读入内存xtree结构（当切分的文件很大需要这样做）。

-N

不显示切分时间和提示。

-D

debug模式。

-U

将闲散单字自动调用二分法结合。

-t *num*

取得前num个高频词。

-a *[~]attr1[,attr2[,...]]*

只显示某些词性的词，加~表示过滤该词性的词，多个词性之间用逗号分隔。

-v

查看版本。

2.1.1.2. scws-gen-dict词典转换工具

执行scws-gen-dict -h可以看到详细帮助说明。

scws-gen-dict ［*option*］［*-i*］*dict.txt* ［*-o*］*dict.xdb*

-c *charset*

指定字符集，默认为gbk，可选utf8。

-i *file*

文本文件(txt)，默认为dict.txt。

-o *file*

输出xdb文件的路径，默认为dict.xdb。

-p *num*

指定 XDB 结构HASH质数（通常不需要）。

-v

查看版本。

文本词典格式为每行一条记录，各行由4个字段组成，字段之间用若干个空格或制表符(\t)分隔。#开头视为注释忽略不计。

表 2.1. 文本词典格式

#<词>	<词频(TF)>	<逆文本频率指数(IDF)>	<词性(北大标注)>
新词条	12.0	2.2	n

除“词”外，其它字段可忽略不写。若忽略，TF和IDF默认值为1.0而词性为"@"。

由于TXT库动态加载（内部监测文件修改时间自动转换成xdb存于系统临时目录），故建议TXT词库不要过大。

删除词做法，请将词性设为“!”，则表示该词设为无效，即使在其它核心库中存在该词也视为无效。

TF意思是词频(Term Frequency)，IDF意思是逆文本频率指数(Inverse Document Frequency)。TF-IDF（term frequency–inverse document frequency）是一种用于信息检索与数据挖掘的常用加权技术，用以评估一字词对于一个文件集或一个语料库中的其中一份文件的重要程度。字词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在语料库中出现的频率成反比下降。TF-IDF加权的各种形式常被搜索引擎应用，作为文件与用户查询之间相关程度的度量或评级。

注意

1. 自定义词典的格式可以是文本TXT，也可以是二进制的XDB格式。XDB格式效率更高，适合大辞典使用。可以使用scws自带的工具scws-gen-dict将文本词典转换为XDB格式；
2. zhparser默认的词典是简体中文，如果需要繁体中文，可以在SCWS中文分词官网下载已经生成好的XDB格式词典。

2.1.2. zhparser

zhparser是一个UXDB中文分词的插件，通过它，可以使UXDB支持中文的全文检索。

以下配置选项用于控制字典加载行为和分词行为，这些选项都不是必须的，默认都为false(即如果没有在配置文件uxsinodb.conf中设置这些选项，则zhparser的行为与将下面的选项设置为false一致)。

忽略所有的标点等特殊符号： zhparser.punctuation_ignore = f
闲散文字自动以二字分词法聚合： zhparser.seg_with_duality = f
将词典全部加载到内存里： zhparser.dict_in_memory = f

短词复合： zhparser.multi_short = f
散字二元复合： zhparser.multi_duality = f
重要单字复合： zhparser.multi_zmain = f
全部单字复合： zhparser.multi_zall = f
例如认为复合等级为7时分词结果最好，则在uxsinodb.conf添加配置：
zhparser.multi_short = true #短词复合： 1
zhparser.multi_duality = true #散字二元复合： 2
zhparser.multi_zmain = true #重要单字复合： 4
zhparser.multi_zall = false #全部单字复合： 8

除了zhparser自带的词典，用户可以增加自定义词典，自定义词典的优先级高于自带的词典。自定义词典的文件必须放在uxdbinstall/dbsql/share/tsearch_data目录中，zhparser根据文件扩展名确定词典的格式类型，.txt扩展名表示词典是文本格式，.xdb扩展名表示这个词典是xdb格式，多个文件使用逗号分隔，词典的分词优先级由低到高，如下所示。

zhparser.extra_dicts = 'dict_extra.txt,mydict.xdb'

> **注意**
>
> zhparser.extra_dicts和zhparser.dict_in_memory两个选项需要在服务器启动前设置（可以在配置文件中修改然后reload，之后新建连接会生效），其他选项可以随时在session中设置生效。
>
> zhparser的选项与scws相关的选项对应，关于这些选项的含义，请参见第 2.1.1.1 节 “scws分词命令行工具”

2.2. 安装与配置

2.2.1. 安装zhparser

2.2.1.1. 载入postgres_adaptor库

1. 功能开启。

   修改配置文件uxsinodb.conf,，加载postgres_adaptor库。

   shared_preload_libraries = 'postgres_adaptor'

2. 重启dbserver。

> **注意**
>
> 如果提示类似ux_*未定义，可以修改UXDB安装目录下的dbsql/share/extension/postgres_adaptor.data文件。增加相应的PG和UXDB对应关系，需要注意的是这些配置项需要保证字典序排序，每个配置项的最大长度为63个字符，最多为2048个配置项。

2.2.1.2. 安装zhparser解析器

uxsql -d uxdb -U uxdb
uxdb=# **CREATE EXTENSION zhparser;**
uxdb=# **\dFp**
List of text search parsers

```
  Schema |   Name   | Description
----------+----------+---------------------
public    | zhparser |
ux_catalog| default  | default word parser
```

新增“zhparser”FTS解析器。

2.2.2. 获得解析器定义的token类型

```
uxdb=# select ts_token_type('zhparser');
 ts_token_type
----------------------------------------
 (97,a,adjective)
 (98,b,"differentiation (qu bie)")
 (99,c,conjunction)
 (100,d,adverb)
 (101,e,exclamation)
 (102,f,"position (fang wei)")
 (103,g,"root (ci gen)")
 (104,h,head)
 (105,i,idiom)
 (106,j,"abbreviation (jian lue)")
 (107,k,head)
 (108,l,"tmp (lin shi)")
 (109,m,numeral)
 (110,n,noun)
 (111,o,onomatopoeia)
 (112,p,prepositional)
 (113,q,quantity)
 (114,r,pronoun)
 (115,s,space)
 (116,t,time)
 (117,u,auxiliary)
 (118,v,verb)
 (119,w,"punctuation (qi ta biao dian)")
 (120,x,unknown)
 (121,y,"modal (yu qi)")
 (122,z,"status (zhuang tai)")
```

表 2.2. token类型

词性缩写	中文名称	描述
a	形容词	取英语形容词adjective的第1个字母。
b	区别词	取汉字“别”的声母。
c	连词	取英语连词conjunction的第1个字母。
d	副词	取adverb的第2个字母，a已有他用。
e	叹词	取英语叹词exclamation的第1个字母。
f	方位词	取汉字“方”的声母。

词性缩写	中文名称	描述
g	词根	绝大多数语素都能作为合成词的“词根”，取汉字“根”的声母。
h	前接成分	取英语head的第1个字母。
i	成语	取英语成语idiom的第1个字母。
j	简称略语	取汉字“简”的声母。
k	后接成分	取英语成语back的第4个字母，b、a、c已有他用。
l	习用语	习用语尚未成为成语，有“临时性”的意思，取“临”的声母。
m	数词	取英语numeral的第3个字母，n、u已有他用。
n	名词	取英语名词noun的第1个字母。
o	拟声词	取英语拟声词onomatopoeia的第1个字母。
p	介词	取英语介词prepositional的第1个字母。
q	量词	取英语quantity的第1个字母。
r	代词	取英语代词pronoun的第2个字母，p已有他用。
s	处所词	取英语space的第1个字母。
t	时间词	取英语time的第1个字母。
u	其他助词	取英语助词auxiliary的第2个字母，a已有他用。
v	动词	取英语动词verb的第1个字母。
w	标点符号	其他标点符号。
x	非语素字	非语素字只是一个符号，字母x通常用于代表未知数、符号。
y	语气词	取汉字“语”的声母。
z	状态词	取汉字“状”的声母的前一个字母。

2.2.3. 配置实例

2.2.3.1. 创建FTS配置

uxdb=# **CREATE TEXT SEARCH CONFIGURATION testzhcfg (PARSER = zhparser);**

2.2.3.2. 添加token映射

uxdb=# **ALTER TEXT SEARCH CONFIGURATION testzhcfg ADD MAPPING FOR a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z WITH simple;**

为防止遗漏造成匹配失败，将所有token类型全部映射。如果有未添加的token类型，将被屏蔽，示例如下。

```
uxdb=# CREATE TEXT SEARCH CONFIGURATION testzhcfg1 (PARSER = zhparser);
uxdb=# ALTER TEXT SEARCH CONFIGURATION testzhcfg1 ADD MAPPING FOR n,v,a,i,e,l
 WITH simple;
uxdb=# SELECT to_tsvector('testzhcfg1','南大 北大 东大 西大');
to_tsvector
----------------------------
'东大':2 '北大':1 '西大':3

uxdb=# SELECT to_tsvector('testzhcfg','南大 北大 东大 西大');
to_tsvector
-------------------------------------
'东大':3 '北大':2 '南大':1 '西大':4

uxdb=# SELECT ts_debug('testzhcfg','南大 北大 东大 西大');
ts_debug
-----------------------------------------
 (j,"abbreviation (jian lue)",南大, {},,)
 (n,noun,北大, {simple},simple,{北大})
 (n,noun,东大, {simple},simple,{东大})
 (n,noun,西大, {simple},simple,{西大})
```

词典使用的是内置的simple词典，仅做小写转换。词典被用来移除停用词并规范化词。一个被成功地规范化的词被称为一个词位。除了提高搜索质量，规范化和移除停用词减小了文档的tsvector表示的尺寸，因而提高了性能。

根据需要可以灵活定义词典和token映射。

2.3. 示例

2.3.1. 解析器示例

```
uxdb=# SELECT * FROM ts_parse('zhparser', 'hello world! 2010年保障房建设在全国范围内获全面启动，从中央到地方纷纷加大了保障房的建设和投入力度。2011年，保障房进入了更大规模的建设阶段。住房城乡建设部党组书记、部长姜伟新去年底在全国住房城乡建设工作会议上表示，要继续推进保障性安居工程建设。');
 tokid |  token
-------+----------
   101 | hello
   101 | world
   117 | !
   101 | 2010
   113 | 年
   118 | 保障
   110 | 房建
   118 | 设在
   110 | 全国
   110 | 范围
......
```

2.3.2. tsvector转化示例

```
uxdb=# SELECT to_tsvector('testzhcfg','南京市长江大桥');
     to_tsvector
-------------------------
 '南京市':1 '长江大桥':2
```

2.3.3. tsquery转化示例

```
uxdb=# SELECT to_tsquery('testzhcfg','保障房资金压力');
        to_tsquery
---------------------------------
 '保障' & '房' & '资金' & '压力'
```

2.3.4. 查询示例

```
uxdb=# SELECT to_tsvector('testzhcfg','中华人民共和国中国');
        to_tsvector
----------------------------------
'中华人民共和国':1 '中国':2
uxdb=# SELECT to_tsvector('testzhcfg','中华人民共和国中国') @@ '中' :: tsquery;
 ?column?
----------
 f
uxdb=# SELECT to_tsvector('testzhcfg','中华人民共和国中国') @@ '人民' :: tsquery;
 ?column?
----------
 f
uxdb=# SELECT to_tsvector('testzhcfg','中华人民共和国中国') @@ '中国' :: tsquery;
?column?
----------
 t
uxdb=# SELECT to_tsvector('testzhcfg','中华人民共和国中国') @@ '中华人民共和国' ::
 tsquery;
?column?
----------
 t
```

2.3.5. 中文全文检索示例

2.3.5.1. 简单查询

查询中可以使用最简单的SELECT * FROM table WHERE to_tsvector('parser_name', field) @@ 'word';来查询field字段分词中带有word一词的数据：

```
uxdb=# CREATE TABLE table_a(name text);
uxdb=# INSERT INTO table_a values('中华人民共和国中国');
uxdb=# INSERT INTO table_a values('我们都是中国人');
uxdb=# SELECT * FROM table_a WHERE to_tsvector('testzhcfg', name) @@ '中国人';
    name
----------------
 我们都是中国人
```

使用 to_tsquery() 方法将句子解析成各个词的组合向量，当然也可以使用&（AND）、|（OR）和!（NOT）符号拼接自己需要的向量；在查询长句时，可以使用 SELECT * FROM table WHERE to_tsvector('parser_name', field) @@ to_tsquery('parser_name','words')。

```
uxdb=# SELECT * FROM table_a WHERE to_tsvector('testzhcfg', name) @@
 to_tsquery('testzhcfg','中国|中国人');
     name
--------------------
 中华人民共和国中国
 我们都是中国人
```

如果想像MySQL的SQL_CALC_FOUND_ROWS语句一样同步返回结果条数，则可以使用SELECT COUNT(*) OVER() AS score FROM table WHERE ...，UXDB会在每一行数据添加score字段存储查询到的总结果条数。

```
uxdb=# SELECT COUNT(*) OVER() AS score FROM table_a WHERE to_tsvector('testzhcfg',
 name) @@ to_tsquery('testzhcfg','中国人');
 score
-------
    1
```

2.3.5.2. 存储分词结果

使用一个字段来存储分词向量，并在此字段上创建索引来更优地使用分词索引。

```
// 添加一个分词字段。
uxdb=# ALTER TABLE table_a ADD COLUMN tsv_column tsvector;
// 将字段的分词向量更新到新字段中。
uxdb=# UPDATE table_a SET tsv_column = to_tsvector('testzhcfg', coalesce(name,''));
// 在新字段上创建GIN索引。
uxdb=# CREATE INDEX idx_gin_zhcn ON table_a USING GIN(tsv_column);
// 创建更新分词触发器。
CREATE TRIGGER trigger_a BEFORE INSERT OR UPDATE ON table_a FOR EACH ROW
 EXECUTE PROCEDURE tsvector_update_trigger( tsv_column, 'testzhcfg', name);
```

这样，再进行查询时就可以直接使用分词结果，例如SELECT * FROM table_a WHERE tsv_column @@ '中国人';。

需要注意，这时候在往表内插入数据的时候，可能会报错，提示指定parser_name的 schema，这时候可以使用\dF命令查看所有text search configuration的参数。

```
uxdb=# \dF
                 文本搜索组态列表
 架构模式 |  名称   |              描述
------------+------------+---------------------------------------
 public     | testzhcfg  |
 ux_catalog | danish     | configuration for danish language
 ux_catalog | dutch      | configuration for dutch language
 ux_catalog | english    | configuration for english language
 ux_catalog | finnish    | configuration for finnish language
 ux_catalog | french     | configuration for french language
 ux_catalog | german     | configuration for german language
 ux_catalog | hungarian  | configuration for hungarian language
 ux_catalog | italian    | configuration for italian language
 ux_catalog | norwegian  | configuration for norwegian language
```

```
ux_catalog | portuguese | configuration for portuguese language
ux_catalog | romanian   | configuration for romanian language
ux_catalog | russian    | configuration for russian language
ux_catalog | simple     | simple configuration
ux_catalog | spanish    | configuration for spanish language
ux_catalog | swedish    | configuration for swedish language
ux_catalog | turkish    | configuration for turkish language
```

> 注意
>
> schema参数，在创建trigger时需要指定schema，如上面，就需要使用public. testzhcfg。

2.4. 问题与解决

词典收录的词毕竟有限，遇到新词可能不识别。不断完善词典可以缓解这个问题，但不能从根本上避免。如下所示。

```
uxdb=# SELECT to_tsvector('testzhcfg','微信');
 to_tsvector
---------------
'信':2 '微':1
uxdb=# SELECT to_tsvector('testzhcfg','微信') @@ '微信' :: tsquery;
 ?column?
----------
 f
```

虽然这个词没有被识别出来，但是只要对tsquery采用相同分词方法，就可以匹配。

```
uxdb=# SELECT to_tsvector('testzhcfg','微信') @@ to_tsquery('testzhcfg','微信');
 ?column?
----------
 t
```

第 3 章 闪回删除

3.1. 概述

闪回删除是借助于插件uxtrashcan来实现UXDB数据库的“回收站”暂存drop的表的功能。

当误删除一个表时，通过执行flashback table 表名 to before drop，使表中的的数据恢复到drop之前的状态。

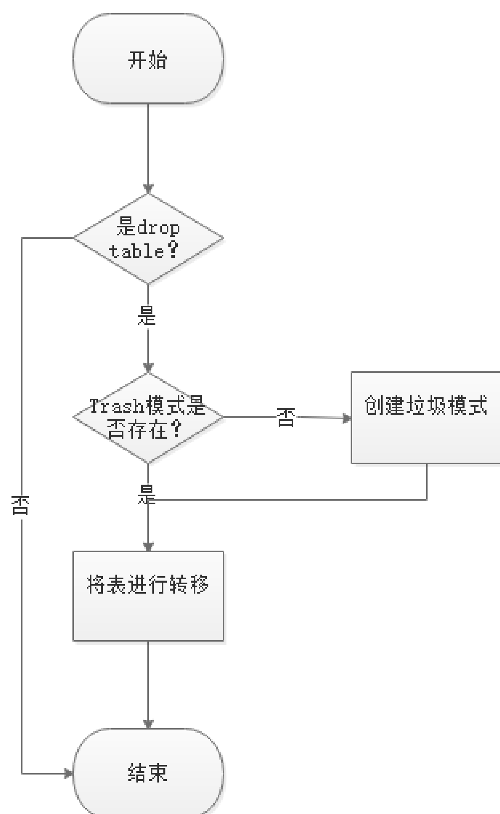
在2.1.1.3版本中，闪回删除不支持Windows系统。

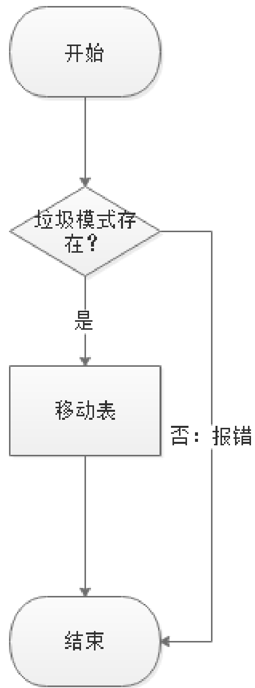
3.2. 原理

闪回删除的实现是基于开源插件uxtrashcan的所提供的垃圾回收容器的特性，使得具有桌面意识的用户在删除表的情况下，可以清晰的知道数据依然存在。主要原理如下所示。

1. 数据库原有的drop table处理逻辑为：创建Trash模式；将表从原public模式下移在Trash模式。

2. 闪回删除的处理逻辑：根据要flashback drop的表名，再反向移动到public模式下。





3.3. 示例

初始化数据库实例。

修改配置文件uxsinodb.conf。

shared_preload_libraries = 'uxtrashcan'

启动数据库实例后支持的语法，如下所示。

- drop table

  语法不变，内部逻辑已变。

  DROP TABLE test1;

- 新增闪回drop

  flashback table test1 to before drop;

1. 创建表t1和表t2，并在表t1插入数据。

```
uxdb=# create table t1(id int);
CREATE TABLE
uxdb=# insert into t1 values(1);
INSERT 0 1
uxdb=# insert into t1 values(2);
INSERT 0 1
uxdb=# create table t2(id int);
CREATE TABLE
```

2. 查看当前系统中已经存在的表，存在表t1和表t2。

```
uxdb=# \d
List of relations
Schema | Name | Type | Owner
-------+------+-------+-------
public | t1 | table | db
public | t2 | table | db
(2 rows)
```

3. 删除表t1并查看。

```
uxdb=# drop table t1;
DROP TABLE
uxdb=# \d
List of relations
Schema | Name | Type | Owner
-------+------+------+-------
public | t2   |table | uxdb
(1 row)
```

4. 删除表t1并查看。

```
uxdb=# drop table t1;
DROP TABLE
uxdb=# \d
List of relations
Schema | Name | Type | Owner
-------+------+------+-------
public | t2   | table| uxdb
(1 row)
```

5. 执行flashback闪回表t1。

```
uxdb=# flashback table t1 to before drop;
FLASHBACK DROP COMPLETE 0
```

6. 通过命令\d查看，表t1已经恢复。

```
uxdb=# \d
List of relations
Schema | Name| Type | Owner
-------+-----+------+-------
public | t1  | table| db
public | t2  | table| db
(2 rows)
```

7. 查看表t1数据，恢复正确。

```
uxdb=# select * from t1;
id
----
1
2
(2 rows)
```

第 4 章 闪回表

4.1. 概述

闪回表是当进行了一些误操作并且执行了commit操作，通过表名和需要恢复到的时间点，使表中的的数据恢复到该时间点之前的状态。

误操作包括insert、update和delete。

在2.1.1.3版本中，闪回表不支持Windows系统。

4.2. 原理

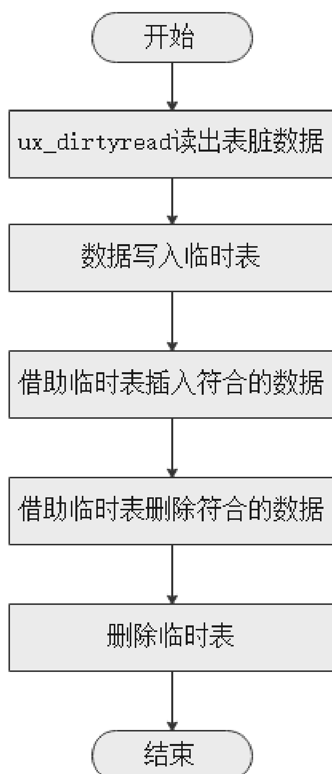
闪回表的实现是基于开源插件ux_dirtyread的所提供的可读脏数据的特性和自定义function实现的。

1. 使用脏读插件ux_dirtyread，将需要flashback的数据（连同头信息xmin,xmax）写入临时表，得到并生成如下字段。

    a. 写入事务提交状态和事务提交时间(xmin)。

    b. 删除事务提交状态和事务提交时间(xmax)。

2. 根据写入临时表的相关字段过滤出当某个timestamp的数据，进行该时刻数据原状态的恢复。



4.3. 示例

修改配置文件uxsinodb.conf。

track_commit_timestamp=on

重启集群后会新增一个闪回表。

```
//闪回至记录的时间
select flashback('t1','2020-09-25 14:37:41');
```

创建extension。

```
uxdb=# create extension ux_dirtyread;
CREATEEXTENSION
```

创建表t1，插入一条数据并查看结果。

```
uxdb=# create table t1(c1 int,c2 text);
CREATETABLE
uxdb=# insert into t1 values(1,'a'),(2,'b');
INSERT 0 2
uxdb=# select * from t1;
c1  |c2
----+----
 1  |a
 2  |b
(2rows)
```

- delete闪回

  1. 记录当前时间。

     ```
     //记录当前时间
     uxdb=# select now();
     now
     -------------------------------
     2020-09-25 14:37:41.385556+08
     (1row)
     ```

  2. 删除表数据并查看结果。

     ```
     uxdb=# delete from t1;
     DELETE2
     uxdb=# select * from t1;
     c1  |c2
     ----+----
     (0rows)
     ```

  3. 闪回至记录的时间，并查看表结果。

     ```
     //闪回至记录的时间
     uxdb=# select flashback('t1','2020-09-2514:37:41');
     flashback
     -----------
     ```

```
(1row)
uxdb=# select * from t1;
c1 | c2
---+----
 2 | b
 1 | a
(2rows)
```

- update闪回

1. 记录当前时间。

```
//记录当前时间
uxdb=# select now();
now
-------------------------------
2020-09-25 14:42:42.366409+08
(1row)
```

2. 更新表数据并查看结果。

```
uxdb=# update t1 set c2='c';
UPDATE2
uxdb=# select * from t1;
 c1 |c2
----+----
 1  |c
 2  |c
(2rows)
```

3. 闪回至记录的时间，并查看表结果。

```
//闪回至记录的时间
uxdb=# select flashback('t1','2020-09-25 14:44:56');
flashback
-----------
(1row)
uxdb=# select * from t1;
c1 |c2
---+----
 1 |a
 2 |b
(2rows)
```

- insert闪回

1. 记录当前时间。

```
//记录当前时间
uxdb=# select now();
now
-------------------------------
2020-09-25 14:44:56.059511+08
(1row)
```

2. 插入一条数据并查看结果。

```
uxdb=# insert into t1 values(3,'c');
INSERT 0 1
uxdb=# select * from t1;
c1 |c2
---+----
 1 |a
 2 |b
 3 |c
(3rows)
```

3. 闪回至记录的时间，并查看表结果。

```
//闪回至记录的时间
uxdb=# select flashback('t1','2020-09-25 14:44:56');
flashback
-----------
(1row)
uxdb=# select * from t1;
 c1|c2
---+----
 1 |a
 2 |b
(2rows)
```

第 5 章 btree_gin

5.1. 概述

btree_gin为int2、int4、int8、float4、float8、timestamp with time zone、timestamp without time zone、time with time zone、time without time zone、date、interval、oid、money、"char"、varchar、text、bytea、bit、varbit、macaddr、macaddr8、inet、cidr数据类型和所有enum类型提供了实现B树等效行为的GIN操作符类样例。

通常，这些操作符类不会比等效的标准B树索引方法更好，并且缺少标准B树代码的一个主要特性：强制唯一性的能力。但是，它们有助于GIN测试并且可以作为开发其他GIN操作符类的基础。另外，对于测试一个GIN可索引的列和一个B树可索引的列的查询，创建一个使用这些操作符类之一的多列GIN索引要比创建必须通过AND组合在一起的两个独立索引要更有效。

5.2. 示例

```
CREATE EXTENSION btree_gin;
CREATE TABLE test (a int4);

-- create index

    CREATE INDEX testidx ON test USING GIN (a);

-- query

    SELECT * FROM test WHERE a < 10;
```

第 6 章 btree_gist

6.1. 概述

btree_gist为int2、int4、int8、float4、float8、numeric、timestamp with time zone、timestamp without time zone、time with time zone、time without time zone、date、interval、oid、money、char、varchar、text、bytea、bit、varbit、macaddr、macaddr8、inet、cidr、uuid数据类型和所有enum类型提供了实现B树等效行为的GiST索引操作符类。

通常，这些操作符类不会比等效的标准B树索引方法更好，并且缺少标准B树代码的一个主要特性：强制唯一性的能力。但是，它们提供了在B树索引中没有的其他特性：当需要一个多列GiST索引，并且其某些列的数据类型只在GiST中是可索引的而其他列是简单数据类型时，可以使用这些操作符类；其次，这些操作符可以用于GiST测试以及作为开发其他GiST操作符类的基础。

除了典型的B树搜索操作符，btree_gist也为<>（“不等于”）提供了索引支持。可与下文描述的排他约束组合在一起产生作用。

另外，对于有自然距离度量的数据类型，btree_gist定义了一个距离操作符<->，并使用这个操作符为最近邻搜索提供了GiST索引支持。距离操作符还提供给了：int2、int4、int8、float4、float8、timestamp with time zone、timestamp without time zone、time without time zone、date、interval、oid和money。

6.2. 示例

CREATE EXTENSION btree_gist;

1. 用btree_gist代替btree，如下所示。

   CREATE TABLE test (a int4);

   -- create *index*

   CREATE INDEX testidx01 ON test USING GIST (a);

   -- query

   SELECT * FROM test WHERE a < 10;

   -- nearest-neighbor search: *find the ten entries closest to "42"*

   SELECT *, a <-> 42 AS dist FROM test ORDER BY a <-> 42 LIMIT 10;

2. 使用一个排他约束来强制规则，一个动物园里的一个笼子只能装一种动物。

   => **CREATE TABLE zoo (**
   cage INTEGER,
   animal TEXT,

```
  EXCLUDE USING GIST (cage WITH =, animal WITH <>)
);
=> INSERT INTO zoo VALUES(123, 'zebra');
INSERT 0 1
=> INSERT INTO zoo VALUES(123, 'zebra');
INSERT 0 1
=> INSERT INTO zoo VALUES(123, 'lion');
ERROR:  conflicting key value violates exclusion constraint "zoo_cage_animal_excl"
DETAIL:  Key (cage, animal)=(123, lion) conflicts with existing key (cage, animal)=(123, zebra).
=> INSERT INTO zoo VALUES(124, 'lion');
INSERT 0 1
```

第 7 章 cstore_fdw

7.1. 概述

cstore_fdw利用列性质仅通过从磁盘读取相关数据来提供性能，并且可以将数据压缩，以减少数据归档的空间需求。

这个扩展使用了ORC（全称Optimized Row Columnar）数据存储格式，ORC改进了Facebook的RCFile格式，有如下优点。

- 压缩：将内存和磁盘中数据大小削减到2到4倍。
- 投影：只提取和查询相关的列数据。提升IO敏感查询的性能。
- 跳过索引：为行组存储最大最小统计值，并利用它们跳过无关的行。

在2.1.1.3版本中，cstore_fdw不支持Windows系统。

在2.1.1.3版本中，UXDB的标准版不支持cstore_fdw。

7.2. 示例

1. 安装uxdb数据库。
2. 设置cstore_fdw的配置，修改集群目录下uxsiondb.conf文件中的选项。

   shared_preload_libraries = 'cstore_fdw'
3. 启动数据库服务。

   ./ux_ctl -D clustername start
4. 连接数据库服务。

   ./uxsql -d uxdb
5. 创建cstore_fdw的extension。

   //load extension first time after install
CREATE EXTENSION cstore_fdw;
6. 创建cstore server。

   //create server object
CREATE SERVER cstore_server FOREIGN DATA WRAPPER cstore_fdw;
7. 创建表。

   CREATE FOREIGN TABLE customer_reviews
(
customer_id TEXT,

review_date DATE,
review_rating INTEGER,
review_votes INTEGER,
review_helpful_votes INTEGER,
product_id CHAR(10),
product_title TEXT,
product_sales_rank BIGINT,
product_group TEXT,
product_category TEXT,
product_subcategory TEXT,
similar_product_ids CHAR(10)[]
)SERVER cstore_server
OPTIONS(compression 'uxlz');

8. COPY数据。

   \COPY customer_reviews FROM 'customer_reviews_1998.csv' WITH CSV;
   \COPY customer_reviews FROM 'customer_reviews_1999.csv' WITH CSV;

9. 查看压缩结果。

```
Type "help" for help.

uxdb=# CREATE EXTENSION cstore_fdw;
CREATE EXTENSION
uxdb=# CREATE SERVER cstore_server FOREIGN DATA WRAPPER cstore_fdw;
CREATE SERVER
uxdb=# CREATE FOREIGN TABLE customer_reviews
uxdb-# (
uxdb(#     customer_id TEXT,
uxdb(#     review_date DATE,
uxdb(#     review_rating INTEGER,
uxdb(#     review_votes INTEGER,
uxdb(#     review_helpful_votes INTEGER,
uxdb(#     product_id CHAR(10),
uxdb(#     product_title TEXT,
uxdb(#     product_sales_rank BIGINT,
uxdb(#     product_group TEXT,
uxdb(#     product_category TEXT,
uxdb(#     product_subcategory TEXT,
uxdb(#     similar_product_ids CHAR(10)[]
uxdb(# )
uxdb-# SERVER cstore_server
uxdb-# OPTIONS(compression 'uxlz');
CREATE FOREIGN TABLE
uxdb=# select * from customer_reviews ;
 customer_id | review_date | review_rating | review_votes | review_helpful_votes | product_id | product_title | product_sales_rank | product
_group | product_category | product_subcategory | similar_product_ids
-------------+-------------+---------------+--------------+----------------------+------------+---------------+--------------------+--------
-------+------------------+---------------------+---------------------
(0 rows)

uxdb=# \COPY customer_reviews FROM 'customer_reviews_1998.csv' WITH CSV;
COPY 589859
uxdb=# \COPY customer_reviews FROM 'customer_reviews_1999.csv' WITH CSV;
COPY 1172645
uxdb=# ANALYZE customer_reviews;
ANALYZE
uxdb=# select * from ux_class where relname='customer_reviews';
```

```
[uxdb@local64 song1]$ ls
base        current_logfiles  log           ux_dynshmem  ux_ident.conf  uxmaster.opts  ux_multixact  ux_replslot  uxsinodb.auto.conf  ux_snapshots  ux_stat_tmp  ux_tblspc    UX_VERSION  ux_xact
cstore_fdw  global            ux_commit_ts  ux_hba.conf  ux_logical     uxmaster.pid   ux_notify     ux_serial    uxsinodb.conf       ux_stat       ux_subtrans  ux_twophase  ux_wal
[uxdb@local64 song1]$ tree cstore_fdw
cstore_fdw
└── 13524
    ├── 16395
    └── 16395.footer

1 directory, 2 files
[uxdb@local64 song1]$ ll cstore_fdw/13524
total 102920
-rw-------. 1 uxdb uxdb 105383393 Oct  9 14:37 16395
-rw-------. 1 uxdb uxdb       239 Oct  9 14:37 16395.footer
[uxdb@local64 song1]$ cd ..
[uxdb@local64 bin]$ ls
clusterdb   customer_reviews_1998.csv  dropuser  reindexdb  ux_archivecleanup  ux_checksums     ux_ctl       ux_dump      uxmaster        ux_resetwal  ux_rman         ux_test_timing  vacuumdb
createdb    customer_reviews_1999.csv  ecux      removedb   ux_basebackup      ux_config        uxdb         ux_dumpall   ux_receivewal   ux_restore   uxsql           ux_upgrade
createuser  dropdb                     initdb    song1      uxbench            ux_controldata   ux_diagnose  ux_isready   ux_recvlogical  ux_rewind    ux_test_fsync   ux_waldump
[uxdb@local64 bin]$ ll customer_reviews_1998.csv
-rw-rw-r--. 1 uxdb uxdb 101299118 Jun  2  2012 customer_reviews_1998.csv
[uxdb@local64 bin]$ ll customer_reviews_1999.csv
-rw-rw-r--. 1 uxdb uxdb 198247156 Jun  2  2012 customer_reviews_1999.csv
[uxdb@local64 bin]$
```

或者使用如下SQL查看：

uxdb=# **create table t1 (id int);**
CREATE TABLE

```
uxdb=# insert into t1 select generate_series(1,10000);
INSERT 0 10000
uxdb=# select ux_relation_size('t1'::regclass);
 ux_relation_size
------------------
      368640
(1 row)
uxdb=# create foreign table cstore_test1(id int) SERVER cstore_server OPTIONS(filename
 '/home/uxdb/Desktop/uxdb/test1/bin/1', compression 'uxlz', stripe_row_count '10000',
 block_row_count '10000');
CREATE FOREIGN TABLE
uxdb=# insert into cstore_test1 select * from t1;
INSERT 0 10000
uxdb=# select cstore_table_size('cstore_test1'::regclass);
 cstore_table_size
-------------------
       41327
(1 row)
```

10. 删除扩展。

```
DROP FOREIGN TABLE customer_reviews;
DROP SERVER cstore_server;
CREATE EXTENSION cstore_fdw;
```

第 8 章 dblink

dblink是一个支持在一个数据库会话中连接到其他UXDB数据库的模块。

表 8.1. dblink功能

功能	描述
dblink_connect	打开一个到远程数据库的持久连接
dblink_connect_u	不安全地打开一个到远程数据库的持久连接
dblink_disconnect	关闭一个到远程数据库的持久连接
dblink	在一个远程数据库中执行一个查询
dblink_exec	在一个远程数据库中执行一个命令
dblink_open	在一个远程数据库中打开一个游标
dblink_fetch	从一个远程数据库中的打开的游标返回行
dblink_close	关闭一个远程数据库中的游标
dblink_get_connections	返回所有打开的命名dblink连接的名称
dblink_error_message	得到在命名连接上的最后一个错误消息
dblink_send_query	发送一个异步查询到远程数据库
dblink_is_busy	检查连接是否正在忙于一个异步查询
dblink_get_notify	在一个连接上检索异步通知
dblink_get_result	得到一个异步查询结果
dblink_cancel_query	在命名连接上取消任何活动查询
dblink_get_pkey	返回一个关系的主键字段的位置和域名称
dblink_build_sql_insert	使用一个本地元组构建一个INSERT语句，将主键字段值替换为提供的值
dblink_build_sql_delete	使用所提供的主键字段值构建一个DELETE语句
dblink_build_sql_update	使用一个本地元组构建一个UPDATE语句，将主键字段值替换为提供的值

请参见uxdb_fdw，它以一种更现代和更加兼容标准的架构提供了相同的功能。

使用dblink模块前，首先需要执行CREATE EXTENSION命令：

CREATE EXTENSION dblink;

8.1. dblink_connect

dblink_connect — 打开一个到远程数据库的持久连接。

8.1.1. 大纲

dblink_connect(text connstr) returns text

dblink_connect(text connname, text connstr) returns text

8.1.2. 描述

dblink_connect()建立一个到远程UXDB数据库的连接。要联系的服务器和数据库通过一个标准的libpq连接串来标识。可以选择将一个名称赋予给该连接。多个命名的连接可以被一次打开，但是未命名连接一次只允许打开一个。连接将会持续直到被关闭或者数据库会话结束。

连接串也可以是一个现存外部服务器的名称。在使用外部服务器时，推荐使用外部数据封装器dblink_fdw，请参见示例 。

8.1.3. 参数

表 8.2. dblink_connect 参数说明

名称	描述
connname	要用于这个连接的名称。如果被忽略，将打开一个未命名连接并且替换掉任何现有的未命名连接。
connstr	libpq-风格的连接信息串，例如 hostaddr=127.0.0.1 port=5432 dbname=mydb user=uxdb password=mypasswd。此外，还可以是一个外部服务器的名称。

8.1.4. 返回值

返回状态总是OK，针对任何错误会被该函数抛出一个错误而不是返回。

8.1.5. 注解

只有系统管理员能够使用dblink_connect来创建无口令认证连接。如果非系统管理员需要这种能力，使用dblink_connect_u()。

选择包含等号的连接名是不明智的，因为这会产生与在其他dblink函数中的连接信息串混淆的风险。

8.1.6. 示例

```
SELECT dblink_connect('myconn', 'hostaddr=192.168.1.84 port=5567 dbname=test user=uxdb
 password=123456');
 dblink_connect
----------------
 OK
(1 row)

-- FOREIGN DATA WRAPPER functionality
-- Note: local connection must require password authentication for this to work properly
-- Otherwise, you will receive the following error from dblink_connect():
-- ----------------------------------------------------------------------
-- ERROR:  password is required
-- DETAIL:  Non-superuser cannot connect if the server does not request a password.
-- HINT:  Target server's authentication method must be changed.
```

```
CREATE SERVER fdtest FOREIGN DATA WRAPPER dblink_fdw OPTIONS (hostaddr
 '127.0.0.1', dbname 'contrib_regression');
CREATE USER regress_dblink_user WITH PASSWORD '123456';
CREATE USER MAPPING FOR regress_dblink_user SERVER fdtest OPTIONS (user
 'regress_dblink_user', password '123456');
GRANT USAGE ON FOREIGN SERVER fdtest TO regress_dblink_user;
GRANT SELECT ON TABLE foo TO regress_dblink_user;
\set ORIGINAL_USER :USER
\c - regress_dblink_user
SELECT dblink_connect('myconn', 'fdtest');
 dblink_connect
----------------
 OK
(1 row)
SELECT * FROM dblink('myconn','SELECT * FROM foo') AS t(a int, b text, c text[]);
 a  | b |       c
----+---+---------------
  0 | a | {a0,b0,c0}
  1 | b | {a1,b1,c1}
  2 | c | {a2,b2,c2}
  3 | d | {a3,b3,c3}
  4 | e | {a4,b4,c4}
  5 | f | {a5,b5,c5}
  6 | g | {a6,b6,c6}
  7 | h | {a7,b7,c7}
  8 | i | {a8,b8,c8}
  9 | j | {a9,b9,c9}
 10 | k | {a10,b10,c10}
(11 rows)
\c - :ORIGINAL_USER
REVOKE USAGE ON FOREIGN SERVER fdtest FROM regress_dblink_user;
REVOKE SELECT ON TABLE foo FROM regress_dblink_user;
DROP USER MAPPING FOR regress_dblink_user SERVER fdtest;
DROP USER regress_dblink_user;
DROP SERVER fdtest;
```

8.2. dblink_connect_u

dblink_connect_u — 不安全地打开一个到远程数据库的持久连接。

8.2.1. 大纲

dblink_connect_u(text connstr) returns text

dblink_connect_u(text connname, text connstr) returns text

8.2.2. 描述

dblink_connect_u()和dblink_connect()一样，不过它将允许非系统管理员使用任意认证方式来连接。

如果远程服务器选择了一种不涉及口令的认证方式，那么可能发生模仿以及后续的扩大权限，因为该会话看起来像由运行UXDB的用户发起的。此外，即使远程服务器不要求一个口令，也可能从

服务器环境提供该口令，例如一个属于服务器用户的~/.uxpass文件。这带来的不只是模仿的风险，而且还有将口令暴露给不可信的远程服务器的风险。因此，dblink_connect_u()最初是用所有从PUBLIC撤销的特权安装的，这让它只能被系统管理员调用。在某些情况中，为dblink_connect_u()授予EXECUTE权限给可信的指定用户是合适的，但是必须小心。也推荐任何属于服务器用户的~/.uxpass文件不能包含任何指定了一个通配符主机名的记录。

详见dblink_connect()。

8.3. dblink_disconnect

dblink_disconnect — 关闭一个到远程数据库的持久连接。

8.3.1. 大纲

dblink_disconnect() returns text

dblink_disconnect(text connname) returns text

8.3.2. 描述

dblink_disconnect()关闭一个之前被dblink_connect()打开的连接。不带参数的形式关闭一个未命名连接。

8.3.3. 参数

表 8.3. dblink_disconnect 参数说明

名称	描述
connname	要被关闭的命名连接的名称。

8.3.4. 返回值

返回值一般是OK，针对任何错误会被该函数抛出一个错误而不是返回。

8.3.5. 示例

```
SELECT dblink_disconnect();
 dblink_disconnect
-------------------
 OK
(1 row)
SELECT dblink_disconnect('myconn');
 dblink_disconnect
-------------------
 OK
(1 row)
```

8.4. dblink

dblink — 在一个远程数据库中执行一个查询。

8.4.1. 大纲

dblink(text connname, text sql [, bool fail_on_error]) returns setof record

dblink(text connstr, text sql [, bool fail_on_error]) returns setof record

dblink(text sql [, bool fail_on_error]) returns setof record

8.4.2. 描述

dblink在一个远程数据库中执行一个查询（通常是一个SELECT，但是也可以是任意返回行的SQL语句）。

当给定两个text参数时，第一个被首先作为一个持久连接的名称进行查找；如果找到，该命令会在该连接上被执行。如果没有找到，第一个参数被视作一个用于dblink_connect的连接信息字符串，并且被指出的连接只是在这个命令的持续期间被建立。

8.4.3. 参数

表 8.4. dblink 参数说明

名称	描述
connname	要使用的连接名。忽略这个参数将使用未命名连接。
connstr	如之前为dblink_connect所描述的一个连接信息字符串。
sql	在远程数据库中执行的SQL查询，例如select * from foo。
fail_on_error	如果为真（忽略时的默认值），那么在连接的远端抛出的错误也会导致本地抛出一个错误。如果为假，远程错误只在本地被报告为NOTICE，并且该函数不返回行。

8.4.4. 返回值

该函数返回查询产生的行。因为dblink能与任何查询一起使用，它被声明为返回record，而不是指定任意特定的列集合。这意味着必须指定在调用的查询中所期待的列集合，否则UXDB将不知道会得到什么。这里是一个示例。

```
SELECT *
  FROM dblink('dbname=mydb password=123456', 'select proname, prosrc from ux_proc')
    AS t1(proname name, prosrc text)
 WHERE proname LIKE 'bytea%';
```

FROM子句的“alias”部分必须指定函数将返回的列名及类型（在一个别名中指定列名实际上是标准SQL语法，但是指定列类型是一种UXDB扩展）。这允许系统在尝试执行该函数之前就理解*将展开成什么，以及WHERE子句中的proname指的什么。在运行时，如果来自远程数据库的实际查询结果和FROM子句中显示的列数不同，将会抛出一个错误。不过，列名不需要匹配，并且dblink并不坚持精确地匹配类型。只要被返回的数据字符串是FROM子句中声明的列类型的合法输入，它就将会成功。

8.4.5. 注解

一种将预定义查询用于dblink的简便方法是创建一个视图。这允许列类型信息被埋藏在该视图中，而不是在每一个查询中都拼写出来。

```
CREATE VIEW myremote_ux_proc AS
SELECT *
 FROM dblink('dbname=uxdb password=123456', 'select proname, prosrc from ux_proc')
   AS t1(proname name, prosrc text);
SELECT * FROM myremote_ux_proc WHERE proname LIKE 'bytea%';
```

8.4.6. 示例

```
SELECT * FROM dblink('dbname=uxdb password=123456', 'select proname, prosrc from
 ux_proc')
   AS t1(proname name, prosrc text) WHERE proname LIKE 'bytea%';
  proname   |   prosrc
------------+------------
 byteacat   | byteacat
 byteaeq    | byteaeq
 bytealt    | bytealt
 byteale    | byteale
 byteagt    | byteagt
 byteage    | byteage
 byteane    | byteane
 byteacmp   | byteacmp
 bytealike  | bytealike
 byteanlike | byteanlike
 byteain    | byteain
 byteaout   | byteaout
(12 rows)
SELECT dblink_connect('dbname=uxdb password=123456');
 dblink_connect
----------------
 OK
(1 row)
SELECT * FROM dblink('select proname, prosrc from ux_proc')
   AS t1(proname name, prosrc text) WHERE proname LIKE 'bytea%';
  proname   |   prosrc
------------+------------
 byteacat   | byteacat
 byteaeq    | byteaeq
 bytealt    | bytealt
 byteale    | byteale
 byteagt    | byteagt
 byteage    | byteage
 byteane    | byteane
 byteacmp   | byteacmp
 bytealike  | bytealike
 byteanlike | byteanlike
 byteain    | byteain
 byteaout   | byteaout
```

```
(12 rows)
SELECT dblink_connect('myconn', 'dbname=regression password=123456');
 dblink_connect
----------------
 OK
(1 row)
SELECT * FROM dblink('myconn', 'select proname, prosrc from ux_proc')
   AS t1(proname name, prosrc text) WHERE proname LIKE 'bytea%';
  proname   |   prosrc
------------+------------
 bytearecv  | bytearecv
 byteasend  | byteasend
 byteale    | byteale
 byteagt    | byteagt
 byteage    | byteage
 byteane    | byteane
 byteacmp   | byteacmp
 bytealike  | bytealike
 byteanlike | byteanlike
 byteacat   | byteacat
 byteaeq    | byteaeq
 bytealt    | bytealt
 byteain    | byteain
 byteaout   | byteaout
(14 rows)
```

8.5. dblink_exec

dblink_exec — 在一个远程数据库中执行一个命令。

8.5.1. 大纲

dblink_exec(text connname, text sql [, bool fail_on_error]) returns text

dblink_exec(text connstr, text sql [, bool fail_on_error]) returns text

dblink_exec(text sql [, bool fail_on_error]) returns text

8.5.2. 描述

dblink_exec在一个远程数据库中执行一个命令（也就是，任何不返回行的SQL语句）。

当给定两个text参数时，第一个被首先作为一个持久连接的名称进行查找；如果找到，该命令会在该连接上被执行。如果没有找到，第一个参数被视作一个用于dblink_connect的连接信息字符串，并且被指出的连接只是在这个命令的持续期间被建立。

8.5.3. 参数

表 8.5. dblink_exec 参数说明

名称	描述
connname	要使用的连接名。忽略这个参数将使用未命名连接。

名称	描述
connstr	如之前为dblink_connect所描述的一个连接信息字符串。
sql	在远程数据库中执行的SQL命令，例如insert into foo values(0,'a','{"a0","b0","c0"}')。
fail_on_error	如果为真（忽略时的默认值），那么在连接的远端抛出的一个错误也会导致本地抛出一个错误。如果为假，远程错误只在本地被报告为一个 NOTICE，并且该函数的返回值被设置为ERROR。

8.5.4. 返回值

返回状态，可能是命令的状态字符串或ERROR。

8.5.5. 示例

```
SELECT dblink_connect('dbname=dblink_test_standby password=123456');
 dblink_connect
----------------
 OK
(1 row)
SELECT dblink_exec('insert into foo values(21,''z'',''{"a0","b0","c0"}'');');
  dblink_exec
-----------------
 INSERT 943366 1
(1 row)
SELECT dblink_connect('myconn', 'dbname=regression password=123456');
 dblink_connect
----------------
 OK
(1 row)
SELECT dblink_exec('myconn', 'insert into foo values(21,''z'',''{"a0","b0","c0"}'');');
  dblink_exec
------------------
 INSERT 6432584 1
(1 row)
```

8.6. dblink_open

dblink_open — 在一个远程数据库中打开一个游标。

8.6.1. 大纲

dblink_open(text cursorname, text sql [, bool fail_on_error]) returns text

dblink_open(text connname, text cursorname, text sql [, bool fail_on_error]) returns text

8.6.2. 描述

dblink_open()在一个远程数据库中打开一个游标。该游标能够随后使用dblink_fetch()和dblink_close()进行操纵。

8.6.3. 参数

表 8.6. dblink_open 参数说明

名称	描述
connname	要使用的连接名。忽略这个参数将使用未命名连接。
cursorname	要赋予给这个游标的名称。
sql	在远程数据库中执行的SELECT语句，例如select * from ux_class。
fail_on_error	如果为真（忽略时的默认值），那么在连接的远端抛出的一个错误也会导致本地抛出一个错误。如果为假，远程错误只在本地被报告为一个NOTICE，并且该函数的返回值被设置为ERROR。

8.6.4. 返回值

返回状态，OK或者ERROR。

8.6.5. 注解

因为一个游标只能在一个事务中持续，如果远端还没有在一个事务中，dblink_open会在远端开始一个显式事务块（BEGIN）。当匹配的dblink_close被执行时，这个事务将再次被关闭。如果使用dblink_exec在dblink_open和dblink_close之间改变数据，并且接着发生了一个错误或者在dblink_close之前使用了dblink_disconnect，更改将被丢失，因为事务将被中止。

8.6.6. 示例

```
SELECT dblink_connect('dbname=uxdb password=123456');
 dblink_connect
----------------
 OK
(1 row)
SELECT dblink_open('foo', 'select proname, prosrc from ux_proc');
 dblink_open
-------------
 OK
(1 row)
```

8.7. dblink_fetch

dblink_fetch — 从一个远程数据库中的打开的游标返回行。

8.7.1. 大纲

dblink_fetch(text cursorname, int howmany [, bool fail_on_error]) returns setof record

dblink_fetch(text connname, text cursorname, int howmany [, bool fail_on_error]) returns setof record

8.7.2. 描述

dblink_fetch从之前由dblink_open建立的游标中取得行。

8.7.3. 参数

表 8.7. dblink_fetch 参数说明

名称	描述
connname	要使用的连接名。忽略这个参数将使用未命名连接。
cursorname	要从中取数据的游标名。
howmany	要检索的最大行数。从当前游标位置向前的接下来howmany个行会被取出。一旦该游标已经到达了它的末端，将不会产生更多行。
fail_on_error	如果为真（忽略时的默认值），那么在连接的远端抛出的一个错误也会导致本地抛出一个错误。如果为假，远程错误只在本地被报告为一个NOTICE，并且该函数不返回行。

8.7.4. 返回值

该函数返回从游标中取出的行。要使用这个函数，需要指定想要的列集合，如前面dblink中所讨论的。

8.7.5. 注解

当FROM子句中指定的返回列的数量和远程游标返回的实际列数不匹配时，将抛出一个错误。在这个事件中，远程游标仍会被前进错误没发生时应该前进的行数。对于远程FETCH完成之后在本地查询中发生的任何其他错误，情况也是一样。

8.7.6. 示例

```
SELECT dblink_connect('dbname=uxdb password=123456');
 dblink_connect
----------------
 OK
(1 row)
SELECT dblink_open('foo', 'select proname, prosrc from ux_proc where proname like ''bytea
%''');
 dblink_open
-------------
 OK
(1 row)
SELECT * FROM dblink_fetch('foo', 5) AS (funcname name, source text);
```

```
 funcname |  source
----------+----------
 byteacat | byteacat
 byteacmp | byteacmp
 byteaeq  | byteaeq
 byteage  | byteage
 byteagt  | byteagt
(5 rows)
SELECT * FROM dblink_fetch('foo', 5) AS (funcname name, source text);
 funcname  |  source
-----------+-----------
 byteain   | byteain
 byteale   | byteale
 bytealike | bytealike
 bytealt   | bytealt
 byteane   | byteane
(5 rows)
SELECT * FROM dblink_fetch('foo', 5) AS (funcname name, source text);
  funcname  |   source
------------+------------
 byteanlike | byteanlike
 byteaout   | byteaout
(2 rows)
SELECT * FROM dblink_fetch('foo', 5) AS (funcname name, source text);
 funcname | source
----------+--------
(0 rows)
```

8.8. dblink_close

dblink_close — 关闭一个远程数据库中的游标。

8.8.1. 大纲

dblink_close(text cursorname [, bool fail_on_error]) returns text

dblink_close(text connname, text cursorname [, bool fail_on_error]) returns text

8.8.2. 描述

dblink_close关闭一个之前由dblink_open打开的游标。

8.8.3. 参数

表 8.8. dblink_close 参数说明

名称	描述
connname	要使用的连接名。忽略这个参数将使用未命名连接。
cursorname	要关闭的游标名。
fail_on_error	如果为真（忽略时的默认值），那么在连接的远端抛出的一个错误也会导致本地抛出一个错

名称	描述
	误。如果为假，远程错误只在本地被报告为一个NOTICE，并且该函数的返回值被设置为ERROR。

8.8.4. 返回值

返回状态，OK或者ERROR。

8.8.5. 注解

如果dblink_open开始了一个显式事务块，并且这是这个连接中最后一个保持打开的游标，dblink_close将发出匹配的COMMIT。

8.8.6. 示例

```
SELECT dblink_connect('dbname=uxdb password=123456');
 dblink_connect
----------------
 OK
(1 row)
SELECT dblink_open('foo', 'select proname, prosrc from ux_proc');
 dblink_open
-------------
 OK
(1 row)
SELECT dblink_close('foo');
 dblink_close
--------------
 OK
(1 row)
```

8.9. dblink_get_connections

dblink_get_connections — 返回所有打开的命名dblink连接的名称。

8.9.1. 大纲

dblink_get_connections() returns text[]

8.9.2. 描述

dblink_get_connections返回一个数组，其中是所有打开的命名dblink连接的名称。

8.9.3. 返回值

返回一个连接名称的文本数组，如果没有则为NULL。

8.9.4. 示例

```
SELECT dblink_get_connections();
```

```
 dblink_get_connections
------------------------
 {myconn}
(1 row)
```

8.10. dblink_error_message

dblink_error_message — 得到命名连接上的最后一个错误消息。

8.10.1. 大纲

dblink_error_message(text connname) returns text

8.10.2. 描述

dblink_error_message为一个给定连接取得最近的远程错误消息。

8.10.3. 参数

表 8.9. dblink_error_message 参数说明

名称	描述
connname	要使用的连接名。

8.10.4. 返回值

返回最后一个错误消息，如果在这个连接上没有错误则返回OK。

8.10.5. 示例

```
SELECT dblink_error_message('dtest1');
 dblink_error_message
----------------------
 OK
(1 row)
SELECT * FROM dblink('myconn', 'select proname, prosrc from ux_proc1')
  AS t1(proname name, prosrc text) WHERE proname LIKE 'bytea%';
错误：关系 "ux_proc1" 不存在
背景： Error occurred on dblink connection named "myconn": could not execute query.
SELECT dblink_error_message('myconn');
  dblink_error_message
------------------------------------------
 错误:  关系 "ux_proc1" 不存在           +
 第1行select proname, prosrc from ux_proc1+
                                  ^
(1 row)
```

8.11. dblink_send_query

dblink_send_query — 发送一个异步查询到远程数据库。

8.11.1. 大纲

dblink_send_query(text connname, text sql) returns int

8.11.2. 描述

dblink_send_query发送一个要被异步执行的查询，也就是不需要立即等待结果。在该连接上不能有还在处理中的异步查询。

在成功地派送一个异步查询后，可以用dblink_is_busy检查完成状态，并且结果最终由dblink_get_result收集。也可以使用dblink_cancel_query尝试取消一个活动中的异步查询。

8.11.3. 参数

表 8.10. dblink_send_query 参数说明

名称	描述
connname	要使用的连接名。
sql	在远程数据库中执行的SQL语句，例如select * from ux_class。

8.11.4. 返回值

如果查询被成功地派送返回1，否则返回0。

8.11.5. 示例

```
SELECT dblink_send_query('dtest1', 'SELECT * FROM foo WHERE f1 < 3');
dblink_send_query
-------------------
                 1
(1 row)
```

8.12. dblink_is_busy

dblink_is_busy — 检查连接是否正在忙于一个异步查询。

8.12.1. 大纲

dblink_is_busy(text connname) returns int

8.12.2. 描述

dblink_is_busy测试是否一个异步查询正在进行中。

8.12.3. 参数

表 8.11. dblink_is_busy 参数说明

名称	描述
connname	要检查的连接名。

8.12.4. 返回值

如果连接正忙则返回1，如果不忙则返回0。如果这个函数返回0，dblink_get_result将被保证不会阻塞。

8.12.5. 示例

```
SELECT dblink_is_busy('dtest1');
dblink_is_busy
----------------
             0
(1 row)
```

8.13. dblink_get_notify

dblink_get_notify — 在一个连接上检索异步通知。

8.13.1. 大纲

dblink_get_notify() returns setof (notify_name text, be_pid int, extra text)

dblink_get_notify(text connname) returns setof (notify_name text, be_pid int, extra text)

8.13.2. 描述

dblink_get_notify在一个未命名连接或者一个指定的命名连接上检索通知。要通过 dblink 接收通知，首先必须使用dblink_exec发出LISTEN。

8.13.3. 参数

表 8.12. dblink_get_notify 参数说明

名称	描述
connname	要得到通知的命名连接的名称。

8.13.4. 返回值

返回 (notify_name text, be_pid int, extra text) 集合，或者一个空集。

8.13.5. 示例

```
SELECT dblink_exec('myconn', 'LISTEN virtual');
 dblink_exec
-------------
 LISTEN
(1 row)
SELECT * FROM dblink_get_notify();
 notify_name | be_pid | extra
-------------+--------+-------
(0 rows)
NOTIFY virtual;
NOTIFY
```

```
SELECT * FROM dblink_get_notify();
 notify_name | be_pid | extra
-------------+--------+-------
 virtual     |   1229 |
(1 row)
```

8.14. dblink_get_result

dblink_get_result — 得到一个异步查询结果。

8.14.1. 大纲

```
dblink_get_result(text connname [, bool fail_on_error]) returns setof record
```

8.14.2. 描述

dblink_get_result收集之前dblink_send_query发送的一个异步查询的结果。如果该查询还没有完成，dblink_get_result将等待直到它完成。

8.14.3. 参数

表 8.13. dblink_get_result 参数说明

名称	描述
connname	要使用的连接名。
fail_on_error	如果为真（忽略时的默认值），那么在连接的远端抛出的一个错误也会导致本地抛出一个错误。如果为假，远程错误只在本地被报告为一个NOTICE，并且该函数不返回行。

8.14.4. 返回值

对于一个异步查询（也就是一个返回行的SQL语句），该函数返回查询产生的行。要使用这个函数，需要指定所期待的列集合，如前面为dblink所讨论的那样。

对于一个异步命令（也就是一个不返回行的SQL语句），该函数返回一个只有单个文本列的单行，其中包含了该命令的状态字符串。仍必须在调用的FROM子句中指定结果将具有一个单一文本行。

8.14.5. 注解

如果dblink_send_query返回1，这个函数就必须被调用。对每一个已发送的查询都必须调用一次这个函数，并且在连接再次可用之前还要多调用一次来得到一个空结果集。

当使用dblink_send_query和dblink_get_result时，在将结果集中的任何一行返回给本地查询处理器之前，dblink将取得整个远程查询结果。如果该查询返回大量的行，这可能会导致本地会话中短暂的内存膨胀。最好将这样的一个查询用dblink_open打开成一个游标并且接着每次取得数量可管理的行。也可以使用dblink()，避免缓冲大型结果集到磁盘上导致的内存膨胀。

8.14.6. 示例

```
SELECT dblink_connect('dtest1', 'dbname=contrib_regression password=123456');
```

```
 dblink_connect
----------------
 OK
(1 row)
SELECT * FROM dblink_send_query('dtest1', 'select * from foo where f1 < 3') AS t1;
 t1
----
  1
(1 row)
SELECT * FROM dblink_get_result('dtest1') AS t1(f1 int, f2 text, f3 text[]);
 f1 | f2 |     f3
----+----+------------
  0 | a  | {a0,b0,c0}
  1 | b  | {a1,b1,c1}
  2 | c  | {a2,b2,c2}
(3 rows)
SELECT * FROM dblink_get_result('dtest1') AS t1(f1 int, f2 text, f3 text[]);
 f1 | f2 | f3
----+----+----
(0 rows)
SELECT * FROM dblink_send_query('dtest1', 'select * from foo where f1 < 3; select * from foo
 where f1 > 6') AS t1;
 t1
----
  1
(1 row)
SELECT * FROM dblink_get_result('dtest1') AS t1(f1 int, f2 text, f3 text[]);
 f1 | f2 |     f3
----+----+------------
  0 | a  | {a0,b0,c0}
  1 | b  | {a1,b1,c1}
  2 | c  | {a2,b2,c2}
(3 rows)
SELECT * FROM dblink_get_result('dtest1') AS t1(f1 int, f2 text, f3 text[]);
 f1 | f2 |      f3
----+----+---------------
  7 | h  | {a7,b7,c7}
  8 | i  | {a8,b8,c8}
  9 | j  | {a9,b9,c9}
 10 | k  | {a10,b10,c10}
(4 rows)
SELECT * FROM dblink_get_result('dtest1') AS t1(f1 int, f2 text, f3 text[]);
 f1 | f2 | f3
----+----+----
(0 rows)
```

8.15. dblink_cancel_query

dblink_cancel_query — 在命名连接上取消任何活动查询。

8.15.1. 大纲

dblink_cancel_query(text connname) returns text

8.15.2. 描述

dblink_cancel_query尝试取消命名连接上正在进行的任何查询。注意这不一定会成功（例如，远程查询可能已经结束）。一个取消请求仅仅提高了该查询将很快失败的几率。仍必须完成常规的查询协议，例如通过调用dblink_get_result。

8.15.3. 参数

表 8.14. dblink_cancel_query 参数说明

名称	描述
connname	要使用的连接名。

8.15.4. 返回值

如果取消请求已经被发送，则返回OK；如果失败，则返回一个错误消息的文本。

8.15.5. 示例

```
SELECT dblink_cancel_query('dtest1');
dblink_cancel_query
---------------------
 OK
(1 row)
```

8.16. dblink_get_pkey

dblink_get_pkey — 返回一个关系的主键字段的位置和字段名称。

8.16.1. 大纲

dblink_get_pkey(text relname) returns setof dblink_pkey_results

8.16.2. 描述

dblink_get_pkey提供有关于本地数据库中一个关系的主键的信息。有助于生成要被发送到远程数据库的查询。

8.16.3. 参数

表 8.15. dblink_get_pkey 参数说明

名称	描述
relname	一个本地关系的名称，例如foo或者myschema.mytab。如果该名称是大小写混合的或包含特殊字符，要使用双引号，例如"FooBar"；如果没有引号，字符串将被转换为小写形式。

8.16.4. 返回值

为每一个主键字段返回一行，如果该关系没有主键则不返回行。结果行类型被定义，如下所示。

CREATE TYPE dblink_pkey_results AS (position int, colname text);

position列值可以从1到N，它是该字段在主键中的编号，而不是在表列中的编号。

8.16.5. 示例

```
CREATE TABLE foobar (
   f1 int,
   f2 int,
   f3 int,
   PRIMARY KEY (f1, f3)
);
CREATE TABLE
SELECT * FROM dblink_get_pkey('foobar');
 position | colname
----------+---------
        1 | f1
        2 | f3
(2 rows)
```

8.17. dblink_build_sql_insert

dblink_build_sql_insert — 使用一个本地元组构建一个INSERT语句，将主键字段值替换为提供的值。

8.17.1. 大纲

dblink_build_sql_insert(text relname, int2vector primary_key_attnums, integer num_primary_key_atts, text[] src_pk_att_vals_array, text[] tgt_pk_att_vals_array) returns text

8.17.2. 描述

dblink_build_sql_insert在选择性地将一个本地表复制到一个远程数据库时很有用。它基于主键从本地表选择一行，并且接着构建一个复制该行的SQL INSERT命令，但是其中主键值被替换为最后一个参数中的值（要创建该行的一个准确拷贝，只要为最后两个参数指定相同的值）。

8.17.3. 参数

表 8.16. dblink_build_sql_insert 参数说明

名称	描述
relname	一个本地关系的名称，例如foo或者myschema.mytab。如果该名称是大小写混合的或包含特殊字符，要使用双引号，例如"FooBar"；如果没有引号，字符串将被转换为小写形式。

名称	描述
primary_key_attnums	主键字段的逻辑列号（从1开始，例如1　2），对应于列在SELECT * FROM relname中的位置。
num_primary_key_atts	主键字段的数量。
src_pk_att_vals_array	要被用来查找本地元组的主键字段值。每一个域都被表示为文本形式。如果没有行具有这些主键值，则抛出一个错误。
tgt_pk_att_vals_array	要被替换到结果INSERT命令中的主键字段值。每一个域被表示为文本形式。

8.17.4.　返回值

将要求的SQL语句返回为文本。

8.17.5.　示例

```
SELECT dblink_build_sql_insert('foo', '1 2', 2, '{"1", "a"}', '{"1", "ba"}');
          dblink_build_sql_insert
--------------------------------------------------
 INSERT INTO foo(f1,f2,f3) VALUES('1','ba','1')
(1 row)
```

8.18.　dblink_build_sql_delete

dblink_build_sql_delete — 使用所提供的主键字段值构建一个DELETE语句。

8.18.1.　大纲

dblink_build_sql_delete(text relname, int2vector primary_key_attnums, integer num_primary_key_atts, text[] tgt_pk_att_vals_array) returns text

8.18.2.　描述

dblink_build_sql_delete在选择性地将一个本地表复制到一个远程数据库时很有用。它构建一个SQL DELETE命令用来删除具有给定主键值的行。

8.18.3.　参数

表　8.17.　dblink_build_sql_delete 参数说明

名称	描述
relname	一个本地关系的名称，例如foo或者myschema.mytab。如果该名称是大小写混合的或包含特殊字符，要使用双引号，例如"FooBar"；如果没有引号，字符串将被转换为小写形式。
primary_key_attnums	主键字段的逻辑列号（从1开始，例如1　2），对应于列在SELECT * FROM relname中的位置。

名称	描述
num_primary_key_atts	主键字段的数量。
tgt_pk_att_vals_array	要用在结果DELETE命令中的主键字段值。每一个字段都被表示为文本形式。

8.18.4. 返回值

将要求的SQL语句返回为文本。

8.18.5. 示例

```
SELECT dblink_build_sql_delete('"MyFoo"', '1 2', 2, '{"1", "b"}');
           dblink_build_sql_delete
---------------------------------------------
 DELETE FROM "MyFoo" WHERE f1='1' AND f2='b'
(1 row)
```

8.19. dblink_build_sql_update

dblink_build_sql_update — 使用一个本地元组构建一个UPDATE语句，将主键字段值替换为提供的值。

8.19.1. 大纲

dblink_build_sql_update(text relname, int2vector primary_key_attnums, integer num_primary_key_atts, text[] src_pk_att_vals_array, text[] tgt_pk_att_vals_array) returns text

8.19.2. 描述

dblink_build_sql_update在选择性地将一个本地表复制到一个远程数据库时很有用。它从本地表基于主键选择一行，并且接着构建一个SQL UPDATE命令来复制该行，但是其中的主键值被替换为最后一个参数中的值（要创建该行的一个准确拷贝，只要为最后两个参数指定相同的值）。UPDATE命令总是为该行的所有字段赋值—这个函数与dblink_build_sql_insert之间的主要区别是它假定目标行已经存在于远程表中。

8.19.3. 参数

表 8.18. dblink_build_sql_update 参数说明

名称	描述
relname	一个本地关系的名称，例如foo或者myschema.mytab。如果该名称是大小写混合的或包含特殊字符，要使用双引号，例如"FooBar"；如果没有引号，字符串将被转换为小写形式。
primary_key_attnums	主键字段的逻辑列号（从1开始，例如1 2），对应于列在SELECT * FROM relname中的位置。
num_primary_key_atts	主键字段的数量。

名称	描述
src_pk_att_vals_array	要被用来查找本地元组的主键字段值。每一个字段都被表示为文本形式。如果没有行具有这些主键值，则抛出一个错误。
tgt_pk_att_vals_array	要用在结果DELETE命令中的主键字段值。每一个字段都被表示为文本形式。

8.19.4. 返回值

将要求的SQL语句返回为文本。

8.19.5. 示例

```
SELECT dblink_build_sql_update('foo', '1 2', 2, '{"1", "a"}', '{"1", "b"}');
               dblink_build_sql_update
-------------------------------------------------------------
 UPDATE foo SET f1='1',f2='b',f3='1' WHERE f1='1' AND f2='b'
(1 row)
```

第 9 章 mysql_fdw

9.1. 概述

uxdb具有插件功能，通过不同的插件拓展实现数据库本身不包含的功能，以满足用户的需求。mysql_fdw 是一个外部表功能，所谓外部表，就是在UXDB数据库中通过SQL访问外部数据源数据，就像访问本地数据库一样；mysql_fdw使用统一的接口方式实现多种数据库的远程访问，包括但不限于MySQL，MongoDB，HDFS等等。

mysql_fdw附带了一个连接池，在第一个使用关联到外部服务器的外部表的查询期间建立一个到外部服务器的连接。这个连接会被保持，并被重用于同一个会话中的后续查询。但是，如果使用了多个用户实体（用户映射）来访问外部服务器，会为每一个用户映射建立一个连接。

mysql_fdw在设置远程访问之后，就可以对映射表进行操作，支持的功能如下所示。

1. 查询操作：支持对外部表的查询操作，即select语句。
2. 写操作：支持对外部表的写操作，即insert、update、delete语句。
3. where子句：外部表上的where条件将在在外部服务器上执行。
4. 导入外部表：支持IMPORT FOREIGN SCHEMA操作。
5. 联合查询：JOIN联合查询，目前，在连接子句中只涉及关系运算符和算术运算符的连接。

> **注意**
>
> 不支持对外表进行CREATE TRIGGER操作。

9.2. 依赖工具

使用mysql_fdw扩展，需要安装mysql客户端工具。

9.3. 示例

1. 安装mysql_fdw扩展。

   uxdb=# **create extension mysql_fdw;**
   CREATE EXTENSION

2. 使用CREATE SERVER创建一个外部服务器。

   示例中连接的MYSQL服务器主机IP：192.71.0.117；端口：3306，如下所示。

   uxdb=# **CREATE SERVER mysql_server FOREIGN DATA WRAPPER mysql_fdw OPTIONS (host '192.71.0.117', port '3306');**
   CREATE SERVER

3. 用CREATE USER MAPPING定义一个用户映射来标识在远程服务器上使用哪个角色。

   示例中将服务器定义映射到uxdb用户上，username为MYSQL的用户名，password为MYSQL用户对应的密码，如下所示。

uxdb=# **CREATE USER MAPPING FOR uxdb SERVER mysql_server OPTIONS (username 'root', password '');**
CREATE USER MAPPING

4. 使用CREATE FOREIGN TABLE创建外部表。

   示例中dbname为MYSQL SERVER中的一个数据库，warehouse为此数据库需要映射的数据表，如下所示。

   uxdb=# **CREATE FOREIGN TABLE warehouse (id int, name text, created timestamp) server mysql_server options (dbname 'mysql', table_name 'warehouse');**
   CREATE FOREIGN TABLE

5. 插入数据到映射表中。

   uxdb=# **INSERT INTO warehouse values (1, 'UPS', current_date);**
   INSERT 0 1
   uxdb=# **INSERT INTO warehouse values (2, 'TV', current_date);**
   INSERT 0 1
   uxdb=# **INSERT INTO warehouse values (3, 'Table', current_date);**
   INSERT 0 1

6. 从表中查询数据。

   uxdb=# **SELECT * FROM warehouse ;**
   ```
    id | name  |       created
   ----+-------+---------------------
     1 | UPS   | 2021-09-02 00:00:00
     2 | TV    | 2021-09-02 00:00:00
     3 | Table | 2021-09-02 00:00:00
   (3 行记录)
   ```

7. 删除一条记录。

   uxdb=# **DELETE FROM warehouse where id = 3;**
   DELETE 1

8. 更新一条数据。

   uxdb=# **UPDATE warehouse set name = 'UPS_NEW' where id = 1;**
   UPDATE 1

第 10 章 oracle_fdw

10.1. 概述

oracle_fdw是一个uxdb扩展，它提供了一个外部数据包装器，用于轻松高效地访问oracle数据库。

oracle_fdw支持如下功能。

- 支持对外部表进行INSERT、UPDATE和DELETE操作。
- 支持可以使用ANALYZE收集外部表的统计信息。
- 自2.1.1.4版本开始，支持IMPORT FOREIGN SCHEMA来批量导入Oracle模式中所有表的表定义。

10.2. 依赖工具

使用oracle_fdw扩展，需要安装oracle客户端工具。

10.3. 选项

10.3.1. 外部数据包装器选项

nls_lang（可选）

将Oracle的NLS_LANG环境变量设置为此值。

NLS_LANG的格式为“language_territory.charset” （例如AMERICAN_AMERICA.AL32UTF8)。这必须与数据库编码相匹配。如果未设置此值，oracle_fdw有可能会自动执行正确的操作，如果不能，则发出警告。

10.3.2. 外部服务器选项

dbserver（必需）

远程数据库的Oracle数据库连接字符串。

只要Oracle客户端进行了相应的配置，这可以是Oracle支持的任何形式。将其设置为本地（“BEQUEATH”）连接的空字符串。

Isolation_level（可选，默认为serializable）

在Oracle数据库中使用的事务隔离级别。该值可以是serializable，read_committed或read_only。

请注意，在单个UXSQL语句期间（例如，在嵌套循环连接期间）可以多次查询Oracle表。为了确保不会发生由并发事务竞争条件引起的不一致，事务隔离级别必须保证读取稳定性。这只能通过Oracle的SERIALIZABLE或READ ONLY隔离级别来保证。

只要Oracle客户端进行了相应的配置，这可以是Oracle支持的任何形式。将其设置为本地（“BEQUEATH”）连接的空字符串。

Oracle对SERIALIZABLE的实现不好，会在意外情况下导致序列化错误(ORA-08177)，例如两个事务并发修改同一个对象，当前一个事务提交或回滚时，第二个事务会收到该错误。

使用READ COMMITTED事务可以解决这个问题，但存在不一致的风险。如果想使用它，请检查执行计划是否可以多次执行外部扫描。

nchar（boolean，可选，默认为off）

将此选项设置为on在Oracle端选择更昂贵的字符转换。如果您使用单字节Oracle数据库字符集，但NCHAR或NVARCHAR2列包含无法在数据库字符集中表示的字符，则这是必需的。

设置nchar为on对性能有显着影响，它会导致UPDATE语句出现ORA-01461错误，这些语句设置的字符串超过2000个字节（如果MAX_STRING_SIZE = EXTENDED，则设置为 16383）。此错误似乎是Oracle错误。

10.3.3. 用户映射选项

User（必需）

会话的Oracle用户名。如果不想将Oracle凭据存储在uxdb数据库中，请将其设置为空字符串以进行外部身份验证（一种简单的方法是使用外部密码存储）。

Password（必需）

Oracle用户的密码。

10.3.4. 外部表选项

table（必需）

Oracle表名。该名称必须与Oracle系统目录中出现的完全相同，因此通常仅由大写字母组成。

要定义基于任意Oracle查询的外部表，请将此选项设置为括号中的查询，例如OPTIONS (table '(SELECT col FROM tab WHERE val = ''string'')')，在这种情况下不要使用schema选项。

INSERT、UPDATE和DELETE将适用于定义在简单查询上的外部表；如果您想避免这种情况（或在更复杂的查询中混淆 Oracle错误消息），请使用表选项readonly。

dblink（可选）

用于访问表的Oracle数据库链接。该名称必须与Oracle系统目录中出现的完全相同，因此通常仅由大写字母组成。

schema（可选）

一般是Oracle用户名，用于访问不属于当前连接Oracle用户的表。该名称必须与Oracle系统目录中出现的完全相同，因此通常仅由大写字母组成。

max_long（可选，默认为32767）

Oracle表中任何LONG、LONG RAW和XMLTYPE 列的最大长度。取值范围是1~1073741823 之间的整数（UXsinoDB中字节的最大长度）。这个内存量至少会分配两次，所以大的值会消耗大量内存。

如果max_long小于检索到的最长值的长度，您将收到错误消息ORA-01406: fetched column value was truncated。

readonly（可选，默认值为“false”）

INSERT、UPDATE和DELETE仅允许在此选项未设置为yes/on/true的表上使用。

sample_percent（可选，默认值为“100”）

此选项仅影响ANALYZE处理，并且可用于在合理时间内分析非常大的表。

该值必须介于0.000001和100之间，并定义将随机选择以计算UXDB表统计信息的Oracle表块的百分比。这是使用SAMPLE BLOCK (x) Oracle 中的子句完成的。

对于使用Oracle查询定义的表，ANALYZE将失败并显示ORA-00933，对于使用复杂Oracle图定义的表可能会失败并显示ORA-01446。

prefetch（可选，默认值为“200”）

设置在外部表扫描期间将通过UXDB和Oracle之间的单次往返获取的行数。这是使用Oracle行预取实现的。该值必须介于0和10240之间，其中值为零将禁用预取。

较高的值可以提高性能，但会在UXDB服务器上使用更多内存。

10.3.5. 列选项

key（可选，默认值是“false”）

如果设置为 yes/on/true，则外部Oracle表上的相应列被视为主键列。

要使UPDATE和DELETE起作用，必须在属于表主键的所有列上设置此选项。

strip_zeros（可选，默认值是“false”）

如果设置为yes/on/true，则传输过程中将从字符串中删除ASCII 0字符。此类字符在Oracle中有效但在UXsinoDB中无效，因此在oracle_fdw 读取时会导致错误。此选项仅对character，character varying和text列有意义。

10.4. 示例

1. 创建oracle_fdw插件。

   进入uxdb数据库执行如下命令。

   uxdb=# **create extension oracle_fdw;**
   CREATE EXTENSION

2. 创建外部连接服务命令。

   127.0.0.1为Oracle本机IP，1521为Oracle端口号，Orcl为Oracle实例名，如下所示。

   uxdb=# **CREATE SERVER osdba_fdw FOREIGN DATA WRAPPER oracle_fdw OPTIONS (dbserver '//127.0.0.1:1521/orcl');**
   CREATE SERVER

3. 创建用户映射，将Oracle端SCOTT用户映射到本地uxdb。

SCOTT为Oracle端用户注意大小写，此大小写与Oracle端一致，Oracle端默认为大写，如下所示。

uxdb=# **CREATE USER MAPPING FOR uxdb SERVER osdba_fdw OPTIONS (user 'SCOTT', password '123456');**
CREATE USER MAPPING

4. 在uxdb创建一个表test_tab，同时在oracle数据库下也创建一个表TEST_TAB。

uxdb=# **CREATE FOREIGN TABLE "test_tab" (id int,name varchar(100)) SERVER osdba_fdw OPTIONS (table 'TEST_TAB');**
CREATE FOREIGN TABLE
SQL> **create table TEST_TAB(id int primary key, name varchar(100));**
Table created.

5. 对映射的表进行操作，操作会反馈到源表上。

对uxdb中的表test_tab插入数据，oracle数据库中表TEST_TAB也会出现插入的数据。

```
uxdb=# INSERT INTO test_tab values (200,'xiaowang'),(300,'ll'),(100,'liutian');
INSERT 0 3
uxdb=# select * from test_tab;
id | name
-----+----------
200 | xiaowang
300 | ll
100 | liutian
(3 行记录)
```

Oracle数据库中可以查到在uxdb中插入的数据，如下所示。

```
SQL> select * from TEST_TAB;
 ID | NAME
----------------
 200 | xiaowang
 300 | ll
 100 | Liutian
(3 行记录)
```

在oracle数据库中插入数据，如下所示。

```
SQL> insert into TEST_TAB values(400,'uu');
1 row created.
SQL> commit 2  ;
Commit complete.
```

在uxdb中查询到在oracle数据库中插入的数据，如下所示。

```
uxdb=# select * from test_tab;
 id |  name
-----+----------
 200 | xiaowang
 300 | ll
 100 | liutian
 400 | uu
```

(4 行记录)

第 11 章 orafce

11.1. 概述

UXDB是和Oracle接近的企业数据库，包括数据类型，功能，架构和语法等几个方面，甚至大多数的日常应用的性能和Oracle一样。

Oracle有些函数或者包，默认UXDB是没有的，需要安装orafce包来实现这些兼容性。

orafce是UXDB的一个extension，主要是为UXDB提供Oracle的部分语法、函数、字典表等兼容。

11.2. 示例

标准模式下，进入uxsql执行create extension orafce。兼容模式下，该插件默认加载，可直接使用。

uxdb=# **create extension orafce;**
CREATE EXTENSIONE

orafce的功能都是基于视图，模式等，部分函数在public模式，部分函数在oracle模式。如果要使用必须得在相应的函数前加上模式名，为了使用方便且和oracle语法保持相似，可以设置search_path参数来避免。有以下三种方式可以设置。

- 全局设置

  在配置文件uxsinodb.conf的“search_path”参数中设置“oracle”和“ux_catalog”。为此，必须在“ux_catalog”之前指定“oracle”。

  search_path='“$user”, public, oracle, ux_catalog, sys';

- 基于数据库级别设置

  ALTER DATABASE db1 SET search_path=“$user”, public, oracle, ux_catalog, sys;

- 基于用户级别设置

  ALTER USER u1 SET search_path=“$user”, public, oracle, ux_catalog, sys;

11.3. 支持特性

orafce支持了Oracle的一千多个函数、十几个系统表、部分数据类型、十几个常用包、部分操作符等。

orafce的实现都是基于函数、视图来实现的。所以如果要做语法兼容，orafce的做法是无法实现的。因为UXDB的语法分析在调用视图和函数之前。必须要在语法分析之前切入hook才能使用extension的实现做语法兼容性。因此，目前的orafce对于Oracle的兼容并不是很完善，契合度不是很高。

11.3.1. 数据类型

支持三种数据类型：oracle.date、varchar2和nvarchar2。

表 11.1. 数据类型

类型名称	对应UXDB类型	Oracle类型
varchar2	varchar	varchar2
nvarchar2	varchar	nvarchar2
oracle.date	timestamp(0)	date

```
uxdb=# create table ora_ux(col1 varchar2,col2 nvarchar2,col3 oracle.date);
CREATE TABLE
uxdb=# insert into ora_ux values ('oracle','uxdb',now());
INSERT 0 1
uxdb=# select * from ora_ux;
  col1  | col2 |        col3
--------+------+---------------------
 oracle | uxdb | 2019-07-10 17:09:16
(1 row)
```

注意

由于UXDB的date类型只有日期没有时间，但Oracle的date类型包含日期和时间，所以orafce将Oracle的date类型用UXDB的timestamp(0)表示，且为了避免与UXDB原生的date类型冲突，使用时需要用oracle.date。

11.3.2. 系统视图和dual表

orafce包含了一些Oracle兼容的系统视图和dual表。

由于存在大量的Oracle用户使用dual表，因此UXDB中创建了dual表，是通过视图来实现的。

表 11.2. 系统视图和dual表

名称	Oracle中该表的内容	orafce中的内容
dual	虚拟表，用来构成select的语法规则	虚拟表，和Oracle一样
oracle.user_tab_columns	保存当前用户的表、视图和Clusters中的列等信息，用于oracle获取表结构	缺少默认值、自增列相关的信息
oracle.user_tables	保存当前用户的关系表，包含表的基本信息和统计信息	只有表名，其他都没有
oracle.user_cons_columns	owner、约束名、表名、列名、position	缺少owner、position
oracle.user_constraints	保存当前用户拥有的所有约束的定义	只有约束名、类型、表名、索引名；缺少搜索条件、owner、删除条件、状态等等
oracle.product_component_version	保存组件的版本、名称、状态信息	同Oracle一样

名称	Oracle中该表的内容	orafce中的内容
oracle.user_objects	保存当前用户拥有的所有对象	缺少时间戳、标记等信息
oracle.user_procedures	保存所有函数和过程及其相关属性	只有对象名称
oracle.user_source	保存当前用户拥有的存储对象的文本源	同Oracle一样
oracle.user_views	保存当前用户拥有的视图信息	只有视图名称和owner，其他都没有
oracle.user_ind_columns	保存当前用户拥有的索引的列	只有表名、索引名、列名，其他都没有
oracle.dba_segments	保存记录各个段的详细信息，包括当前对象所拥有的分配给所有段的存储空间	只有owner、段名、段类型、所属表空间、头文件ID、blockID、段大小、块大小

```
uxdb=# \dv oracle.*
              List of relations
 Schema |          Name           | Type | Owner
--------+---------------------------+------+-------
 oracle | dba_segments              | view | uxdb
 oracle | product_component_version | view | uxdb
 oracle | user_cons_columns         | view | uxdb
 oracle | user_constraints          | view | uxdb
 oracle | user_ind_columns          | view | uxdb
 oracle | user_objects              | view | uxdb
 oracle | user_procedures           | view | uxdb
 oracle | user_source               | view | uxdb
 oracle | user_tab_columns          | view | uxdb
 oracle | user_tables               | view | uxdb
 oracle | user_views                | view | uxdb
(11 rows)
uxdb=# \dv
List of relations
 Schema | Name | Type | Owner
--------+------+------+-------
 public | dual | view | uxdb
(1 row)
```

11.3.3. 函数

UXDB中，系统表ux_proc用来保存函数或存储过程的信息，对比安装orafce前后字典表中的数据行数，可以知道orafce创建了大约400多个函数。

```
uxdb=# select count(*) from ux_proc ;
 count
-------
  2894
(1 row)
uxdb=# create extension orafce;
CREATE EXTENSION
```

```
uxdb=# select count(*) from ux_proc ;
 count
-------
  3384
(1 row)
```

1. 字符串处理：填充和截取

- 字符串填充

```
uxdb=# select oracle.rpad('abcd',8,'tx');
 rpad
----------
 abcdtxtx
(1 row)
uxdb=# select oracle.lpad('abcd',8,'tx');
 lpad
----------
 txtxabcd
(1 row)
```

- 字符串截取

```
uxdb=# select oracle.substr('adbc',2,2);
 substr
--------
 db
(1 row)
uxdb=# SELECT btrim('abcd','cd') FROM dual;
 btrim
-------
 ab
(1 row
```

2. 获取数据库信息

- 返回服务器时间

```
uxdb=# select oracle.sysdate() from dual;
       sysdate
---------------------
 2019-07-11 07:06:48
(1 row)
```

- 返回服务器时区

```
uxdb=# select oracle.dbtimezone() from dual;
 dbtimezone
------------
 GMT
(1 row)
```

- 返回当前会话的时区

```
uxdb=# select oracle.sessiontimezone() from dual;
 sessiontimezone
```

```
-----------------
 PRC
(1 row)
```

- 获取数据库版本

```
uxdb=# select oracle.get_major_version();
 get_major_version
-------------------
 UXsinoDB 10.0
(1 row)
```

3. 日期处理：日期转换、加减、日期范围区间判断

- 返回日期加n个月

```
uxdb=# select oracle.add_months(oracle.date'2019-05-21 10:12:12',1) from dual;
    add_months
---------------------
 2019-06-21 10:12:12
(1 row)
```

- 返回日期值月的最后一天

```
uxdb=# select oracle.last_day(oracle.date '2019-05-21 11:12:12') from dual;
 last_day
---------------------
 2019-05-31 11:12:12
(1 row)
```

- 返回大于日期值的第一个星期日期（可用1-7表示）

```
uxdb=# select oracle.next_day(oracle.date '2019-05-21 10:12:12', 'monday') from dual;
     next_day
---------------------
 2019-05-27 10:12:12
(1 row)
```

- 返回date1和date2之间的月数，如果不是整月，按每月31天算

```
uxdb=# select oracle.months_between(oracle.date '2019-06-21 10:00:00', oracle.date
 '2019-05-21 10:21:11') from dual;
 months_between
----------------
              1
(1 row)
```

4. 类型转换：字符串、日期、数值之间转换

- 日期转换为字符串

```
uxdb=# select oracle.to_char(to_date('14-Jan-19 11:44:49+05:30'));
  to_char
---------------------
 2019-01-14 11:44:49
(1 row)
```

- 字符串转换为日期

```
uxdb=# select oracle.to_date('05/16/19 04:12:12') from dual;
  to_date
---------------------
 2019-05-16 04:12:12
(1 row)
```

11.3.4. 操作符

orafce重载了+ - 操作符，用于支持oracle.date类型与smallint、integer类型的加减。

```
uxdb=# select ux_catalog.to_date('2019-07-02 10:08:55','YYYY-MM-DD HH:MI:SS') +
 9::smallint;
  ?column?
------------
 2019-07-11
(1 row))
uxdb=# select ux_catalog.to_date('2019-07-02 10:08:55','YYYY-MM-DD HH:MI:SS') - 9::integer;
  ?column?
------------
 2019-06-23
(1 row)
uxdb=# select oracle.to_date('2019-07-17 11:10:15', 'yyyy-mm-dd hh24:mi:ss') -
 oracle.to_date('2019-02-01 10:00:00', 'yyyy-mm-dd hh24:mi:ss');
     ?column?
-------------------
 166 days 01:10:15
(1 row)
```

11.3.4.1. oracle.date/float操作符

11.3.4.1.1. 概述

oracle.date与float之间增加了“+”或“-”两种运算操作符。“+”操作符对应的函数为add_days_to_timestamp，“-”操作符对应的函数为sub_days_to_timestamp。

add_days_to_timestamp(oracle.date,float)是orafce插件中对add_days_to_timestamp函数的补充；sub_days_to_timestamp(oracle.date,float)是此次新增加的函数。功能是让这两种类型数据进行相应的加减操作，并得到正确的结果。

11.3.4.1.2. 用法示例

执行如下命令，查看返回结果。

```
uxdb=# select to_date('07-02-2014 10:11:00', 'MM-DD-YYYY HH:MI:SS') + 1.5::float;
      ?column?
---------------------
 2014-07-03 12:00:00
(1 row)
uxdb=# select to_date('07-02-2014 10:11:00', 'MM-DD-YYYY HH:MI:SS') - 1.5::float;
      ?column?
---------------------
 2014-06-30 12:00:00
```

```
(1 row)
uxdb=# select 1.5::float + to_date('07-02-2014 10:11:00', 'MM-DD-YYYY HH:MI:SS');
 ?column?
---------------------
 2014-07-03 12:00:00
(1 row)
```

> **注意**
>
> 不支持float - oracle.date操作。

11.3.5. 支持的包

在UXDB里用schema+函数的形式来实现Oracle兼容包。

```
uxdb=# \dn
 List of schemas
    Name     | Owner
-------------+-------
 dbms_alert   | uxdb
 dbms_assert  | uxdb
 dbms_output  | uxdb
 dbms_pipe    | uxdb
 dbms_random  | uxdb
 dbms_utility | uxdb
 oracle       | uxdb
 plunit       | uxdb
 plvchr       | uxdb
 plvdate      | uxdb
 plvlex       | uxdb
 plvstr       | uxdb
 plvsubst     | uxdb
 public       | uxdb
 utl_file     | uxdb
(15 rows)
```

查看包，例如dbms_output包：

```
uxdb=# \df dbms_output.*
          List of functions
  Schema    |     Name     | Result data type |          Argument data types           |  Type
-------------+--------------+------------------+----------------------------------------+--------
 dbms_output | disable      | void             |                                        | normal
 dbms_output | enable       | void             |                                        | normal
 dbms_output | enable       | void             | buffer_size integer                    | normal
 dbms_output | get_line     | record           | OUT line text, OUT status integer      | normal
 dbms_output | get_lines    | record           | OUT lines text[], INOUT numlines integer | normal
 dbms_output | new_line     | void             |                                        | normal
 dbms_output | put          | void             | a text                                 | normal
 dbms_output | put_line     | void             | a text                                 | normal
 dbms_output | serveroutput | void             | boolean                                | normal
(9 rows)
```

1. dbms_output

UXDB通过raise notice向客户端发送信息。

//打开serveroutput
select dbms_output.serveroutput('t');
//打开dbms_output生效，默认打开
select dbms_output.enable();
//写入buffer但不输出
select dbms_output.put('a');
//写入buffer但不输出
select dbms_output.put('b');
//输出并换行
select dbms_output.put_line('c');

2. utl_file

允许UXSQL prgrams从服务器读写任何文件。每个会话最多可以打开10个文件，最大行大小为32K。

3. dbms_pipe

用于在不同会话之间进行通信。可以建立公有和私有管道，所有用户都可以访问公有管道，只有建立管道的用户可以访问私有管道。

表 11.3. dbms_pipe功能表

功能名称	功能描述
pack_message	用于将消息写入到本地消息缓冲区
send_message	用于将本地消息缓冲区中的内容发送到管道
receive_message	用于接收管道消息
next_item_type	用于确定本地消息缓冲区下一项的数据类型
	如果该返回0,则表示管道没有任何消息
	如果返回6,则表示下一项的数据类型为number
	如果返回9,则表示下一项的数据类型为varchar2
	如果返回11,则表示下一项的数据类型为rowid
	如果返回12,则表示下一项的数据类型为date
	如果返回13,则表示下一项的数据类型为timestamp
	如果返回23,则表示下一项的数据类型为bytea（Oracle中是raw）
unpack_message	用于将消息缓冲区的内容写入到变量中
remove_pipe	用于删除已经建立的管道
puger	用于清除管道中的内容
reset_buffer	用于复位管道缓冲区
unique_session_name	用于为特定会话返回惟一名称，且名称的最长度为30字节

// Session A
//创建一个公用管道
select dbms_pipe.create_pipe('my_pipe',10,true);
//将消息写入到本地消息缓冲区
select dbms_pipe.pack_message('uxsino');
//将消息写入到本地消息缓冲区
select dbms_pipe.pack_message('anything is else');
//本地消息发送到管道
select dbms_pipe.send_message('my_pipe');
//查看管道列表
select * from dbms_pipe.db_pipes;

// Session B
//接收管道消息
select dbms_pipe.receive_message('my_pipe');
//确定本地消息缓冲区下一项的数据类型
select dbms_pipe.next_item_type();
//将消息缓冲区的内容写入到变量中
select dbms_pipe.unpack_message_text();
select dbms_pipe.next_item_type();
//删除已经建立的管道
select dbms_pipe.remove_pipe('my_pipe');

注意

dbms_pipe和Oracle中的不同之处：

1.管道的限制不是以字节为单位的，而是以管道中的元素为单位的。

2.发送消息可以不用等待。

3.可以发送空消息。

4.next_item_type有timestamp类型，返回值是13。

5.UXDB无法识别出raw类型，用bytea代替。

4. dbms_alert

进程间通信的一种方法。用于生成并传递数据库预警信息。

表 11.4. dbms_alert功能表

功能名称	功能描述
dbms_alter.register	用于注册预警事件
dbms_alter.remove	用于删除会话不需要的预警事件
dbms_alter.removeall	用于删除当前会话所有已注册的预警事件
dbms_alter.signal	用于指定预警事件所对应的预警消息
dbms_alter.waitany	用于等待当前会话的任何预警事件，并且在预警事件发生时输出相应信息.在执行该过程之前,会隐含地发出COMMIT（ 注:status用于

功能名称	功能描述
	返回状态值,返回0表示发生了预警事件,返回1表示超时;timeout用于设置预警事件的超时时间)
dbms_alter.waitone	用于等待当前会话的特定预警事件,并且在发生预警事件时输出预警消息

// Session A
//注册预警事件
select dbms_alert.register('uxsino');
//等待预警事件，超时时间是600s
select * from dbms_alert.waitany(600);

// Session A
//在超时时间内执行，则sessionA返回状态为0，否则返回为1
select dbms_alert.signal('uxsino','Nice day');

5. PLVdate

包含了一些工作日的计算，默认配置适用于欧洲国家。

表 11.5. PLVdate功能表

功能名称	功能描述
plvdate.add_bizdays(day date, days int) date	返回date+n个工作日的日期
plvdate.nearest_bizday(day date) date	返回给定日期最近的工作日或休息日
plvdate.next_bizday(day date) date	返回给定日期下一个的工作日或休息日
plvdate.bizdays_between(day1 date, day2 date) int	两个日期之间的工作日数
plvdate.prev_bizday(day date) date	返回给定日期前一个的工作日或休息日
plvdate.isbizday(date) bool	确定是否是工作日
plvdate.set_nonbizday(dow varchar)	将一周中的某一天设为非工作日
plvdate.unset_nonbizday(dow varchar)	将一周中的某一天设置为工作日
plvdate.set_nonbizday(day date)	将当天设为非工作日
plvdate.unset_nonbizday(day date)	将当天设为工作日
plvdate.set_nonbizday(day date, repeat bool)	将给定日期设定为工作日，如果为真，则每年的这个日期都是工作日
plvdate.unset_nonbizday(day date, repeat bool)	将给定日期设定为非工作日，如果为真，则每年的这个日期都是非工作日
plvdate.use_easter()	设置复活节和复活节后的一周为放假日
plvdate.using_easter() bool	如果是复活节则返回true
plvdate.use_great_friday()	复活节后的周五设置为放假日
plvdate.using_easter() bool	如果复活节是周五，则返回真
plvdate.include_start()	在bizdays_between计算中包含开始日期
plvdate.noinclude_start()	在bizdays_between计算中不包含开始日期

功能名称	功能描述
plvdate.default_holidays(varchar)	加载默认配置
plvdate.days_inmonth(date)	返回一个月中的天数
plvdate.isleapyear(date)	确定是否处于闰年

配置只包含所有区域的公共假日。可以使用set_nonbizday自定义区域假日。

6. PLVstr and PLVchr

包含一些字符串和字符相关函数。支持正向偏移和负向偏移。

表 11.6. PLVstr and PLVchr功能表

功能名称	功能描述
plvstr.normalize(str text)	格式化字符串，去除字符串前面多余的空格
plvstr.is_prefix(str text, prefix text, cs bool)	str的前缀是prefix，则返回真
plvstr.is_prefix(str text, prefix text)	str的前缀是prefix，则返回真
plvstr.is_prefix(str int, prefix int)	str的前缀是prefix，则返回真
plvstr.is_prefix(str bigint, prefix bigint)	str的前缀是prefix，则返回真
plvstr.substr(str text, start int, len int)	返回str字符串中从第start位开始的len位字符
plvstr.substr(str text, start int)	返回str字符串中从第start位开始到结束的字符
plvstr.instr(str text, patt text, start int, nth int)	字符串检索
plvstr.instr(str text, patt text, start int)	字符串检索
plvstr.instr(str text, patt text)	字符串检索
plvstr.lpart(str text, div text, start int, nth int, all_if_notfound bool)	all_if_notfound为t，则返回全部;all_if_notfound为f，则返回空;all_if_notfound不定义，则从第start开始，往后nth个字符串中包含div，则返回nth左边的字符
plvstr.lpart(str text, div text, start int, nth int)	从第start开始，往后nth个字符串中包含div，则返回nth左边的字符
plvstr.lpart(str text, div text, start int)	从第start开始，往后的字符串中包含div，则返回start左边的字符
plvstr.lpart(str text, div text)	返回str中div左边的字符
plvstr.rpart(str text, div text, start int, nth int, all_if_notfound bool)	all_if_notfound为t，则返回全部;all_if_notfound为f，则返回空;all_if_notfound不定义，则从第start开始，往后nth个字符串中包含div，则返回nth右边的字符

功能名称	功能描述
plvstr.rpart(str text, div text, start int, nth int)	从第start开始，往后nth个字符串中包含div，则返回nth右边的字符
plvstr.rpart(str text, div text, start int)	从第start开始，往后的字符串中包含div，则返回nth右边的字符
plvstr.rpart(str text, div text)	返回str中div左边的字符
plvstr.lstrip(str text, substr text, num int)	从str最左边开始删除连续的小于等于num个substr字符
plvstr.lstrip(str text, substr text)	从str最左边开始删除一个substr字符
plvstr.rstrip(str text, substr text, num int)	从str最右边开始删除连续的小于等于num个substr字符
plvstr.rstrip(str text, substr text)	从str最右边开始删除一个substr字符
plvstr.rvrs(str text, start int, end int)	返回反转的str中start到end的字符串
plvstr.rvrs(str text, start int)	返回反转的str中start开始的字符串
plvstr.rvrs(str text)	反转str字符串
plvstr.left(str text, n int)	返回第一个到第n个字符串
plvstr.right(str text, n int)	返回第n个到最后一个字符串
plvstr.swap(str text, replace text, start int, lengh int)	用replace替换str中start开始lengh长度的字符串
plvstr.swap(str text, replace text)	用replace替换str中开头的字符串
plvstr.betwn(str text, start int, _end int, inclusive bool)	找到开始和结束位置之间的子字符串
plvstr.betwn(str text, start text, _end text, startnth int, endnth int, inclusive bool, gotoend bool)	找到开始和结束位置之间的子字符串
plvstr.betwn(str text, start text, _end text)	找到开始和结束位置之间的子字符串
plvstr.betwn(str text, start text, _end text, startnth int, endnth int)	找到开始和结束位置之间的子字符串
plvchr.nth(str text, n int)	返回字符串中的第n个字符
plvchr.first(str text)	返回字符串中的第一个字符
plvchr.last(str text)	返回字符串中的最后一个字符
plvchr.is_blank(c int)	是否为空
plvchr.is_blank(c text)	是否为空
plvchr.is_digit(c int)	是否为十进制数字
plvchr.is_digit(c text)	是否为十进制数字
plvchr.is_quote(c int)	是否为特殊字符
plvchr.is_quote(c text)	是否为特殊字符
plvchr.is_other(c int)	--
plvchr.is_other(c text)	--
plvchr.is_letter(c int)	是否是字母

功能名称	功能描述
plvchr.is_letter(c text)	是否是字母
plvchr.quoted1(str text)	引用单引号
plvchr.quoted2(str text)	引用双引号
plvchr.stripped(str text, char_in text)	删除str中的char_in

```
// ab
select plvstr.left('abcdef',2);
// abcd
select plvstr.left('abcdef',-2);
// a
select plvstr.substr('abcdef',1,1);
// f
select plvstr.substr('abcdef',-1,1);
// d
select plvstr.substr('abcde',-2,1);
```

7. DBMS_utility

表 11.7. DBMS_utility功能表

功能名称	功能描述
dbms_utility.format_call_stack()	返回带有调用堆栈内容的格式化字符串

```
uxdb=# select dbms_utility.format_call_stack();
      format_call_stack
---------------------------------
 ----- PL/uxSQL Call Stack -----+
   object     line  object      +
   handle   number  name        +
(1 row)
```

8. PLVlex

与Oracle的PLVlex不兼容。

```
uxdb=# select * from plvlex.tokens('select * from a.b.c join d ON x=y', true, true);
 pos | token  | code |  class  | separator | mod
-----+--------+------+---------+-----------+------
   0 | select |  597 | KEYWORD |           |
   7 | *      |   42 | OTHERS  |           | self
   9 | from   |  417 | KEYWORD |           |
  14 | a.b.c  |      | IDENT   |           |
  20 | join   |  464 | KEYWORD |           |
  25 | d      |      | IDENT   |           |
  27 | on     |  521 | KEYWORD |           |
  30 | x      |      | IDENT   |           |
  31 | =      |   61 | OTHERS  |           | self
  32 | y      |      | IDENT   |           |
(10 rows)
```

9. DBMS_ASSERT

保护用户输入不受SQL注入的影响。

表 11.8. DBMS_ASSERT功能表

功能名称	功能描述
dbms_assert.enquote_literal(varchar) varchar	添加前引号和后引号，验证所有单引号都与相邻的单引号配对
dbms_assert.enquote_name(varchar [, boolean]) varchar	大写的字符串用双引号标记
dbms_assert.noop(varchar) varchar	返回值而不进行任何检查
dbms_assert.qualified_sql_name(varchar) varchar	此函数验证输入字符串是否是限定的SQL名称
dbms_assert.schema_name(varchar) varchar	函数验证输入字符串是否是现有schema名
dbms_assert.simple_sql_name(varchar) varchar	这个函数验证输入字符串是否是简单的SQL名称
dbms_assert.object_name(varchar) varchar	验证输入字符串是否为现有SQL对象的限定SQL标识符

10. PLUnit

包含一些断言函数。

表 11.9. PLUnit功能表

功能名称	功能描述
plunit.assert_true(bool [, varchar])	断言条件为true
plunit.assert_false(bool [, varchar])	断言条件为false
plunit.assert_null(anyelement [, varchar])	断言实际值为空
plunit.assert_not_null(anyelement [, varchar])	断言实际值不为空
plunit.assert_equals(anyelement, anyelement [, double precision] [, varchar])	断言期望和实际是相等的
plunit.assert_not_equals(anyelement, anyelement [, double precision] [, varchar])	断言期望和实际是不相等的
plunit.fail([varchar])	立即失败，并输出消息

11. DBMS_random

用于生成随机数。

表 11.10. DBMS_random功能表

功能名称	功能描述
dbms_random.initialize(int)	初始化一个种子值的包

功能名称	功能描述
dbms_random.normal()	返回标准正态分布中的随机数
dbms_random.random()	返回一个随机值，范围是-2的31次幂到2的31次幂
dbms_random.seed(int)	用于生成一个随机数种子,设置种子的目的是可以重复生成随机数
dbms_random.seed(text)	重置种子值
dbms_random.string(opt text(1), len int)	创建随机字符串
dbms_random.terminate()	终止包 (UXDB中没有任何作用)
dbms_random.value()	在[0.0 - 1.0)之间返回一个随机值
dbms_random.value(low double precision, high double precision)	在[low - high)之间返回一个随机值

11.3.6. 兼容Oracle数据库功能

orafce支持Oracle使用包括sys用户和拥有DBA角色的视图（10个以上）；简洁条件判断函数decode。

11.3.6.1. sys用户和DBA角色视图

支持Oracle中sys用户和拥有DBA角色的视图10个以上，并详细记录视图类型。

查看sys模式下的视图。

```
uxdb=# \dv sys.*
            List of relations
 Schema |       Name        | Type | Owner
--------+-------------------+------+-------
 sys    | all_all_tables    | view | uxdb
 sys    | all_cons_columns  | view | uxdb
 sys    | all_constraints   | view | uxdb
 sys    | all_ind_columns   | view | uxdb
 sys    | all_indexes       | view | uxdb
 sys    | all_objects       | view | uxdb
 sys    | all_sequences     | view | uxdb
 sys    | all_tab_columns   | view | uxdb
 sys    | all_tables        | view | uxdb
 sys    | all_triggers      | view | uxdb
 sys    | all_types         | view | uxdb
 sys    | all_view_columns  | view | uxdb
 sys    | all_views         | view | uxdb
 sys    | dba_all_tables    | view | uxdb
 sys    | dba_cons_columns  | view | uxdb
 sys    | dba_constraints   | view | uxdb
 sys    | dba_ind_columns   | view | uxdb
 sys    | dba_indexes       | view | uxdb
 sys    | dba_objects       | view | uxdb
 sys    | dba_role_privs    | view | uxdb
 sys    | dba_roles         | view | uxdb
 sys    | dba_sequences     | view | uxdb
 sys    | dba_tab_columns   | view | uxdb
```

```
 sys    | dba_tables        | view | uxdb
 sys    | dba_triggers      | view | uxdb
 sys    | dba_types         | view | uxdb
 sys    | dba_view_columns  | view | uxdb
 sys    | dba_views         | view | uxdb
 sys    | user_all_tables   | view | uxdb
 sys    | user_cons_columns | view | uxdb
 sys    | user_constraints  | view | uxdb
 sys    | user_ind_columns  | view | uxdb
 sys    | user_indexes      | view | uxdb
 sys    | user_objects      | view | uxdb
 sys    | user_role_privs   | view | uxdb
 sys    | user_sequences    | view | uxdb
 sys    | user_tab_columns  | view | uxdb
 sys    | user_tables       | view | uxdb
 sys    | user_triggers     | view | uxdb
 sys    | user_types        | view | uxdb
 sys    | user_view_columns | view | uxdb
 sys    | user_views        | view | uxdb
(42 rows)
```

查看视图字段。

```
uxdb=# \d all_all_tables
                    View "sys.all_all_tables"
     Column      |        Type         | Collation | Nullable | Default
-----------------+---------------------+-----------+----------+---------
 owner           | public.varchar2(30) | C         |          |
 schema_name     | public.varchar2(30) | C         |          |
 table_name      | public.varchar2(30) | C         |          |
 tablespace_name | public.varchar2(30) | C         |          |
 status          | public.varchar2(8)  |           |          |
 temporary       | public.varchar2(1)  |           |          |
```

查看视图内容。

```
uxdb=# select * from all_all_tables;
 owner | schema_name |  table_name  | tablespace_name | status | temporary
-------+-------------+--------------+-----------------+--------+-----------
 uxdb  | public      | student      |                 | VALID  | N
 uxdb  | utl_file    | utl_file_dir |                 | VALID  | N
(2 rows)
```

11.3.6.2. 简洁条件判断函数decode

支持简洁条件判断函数decode。

1. decode(integer, integer, text, integer, text, integer, text, integer, text)
2. decode(integer, integer, text, integer, text, integer, text, integer, text, integer, text)
3. create function decode(text, varchar2)

示例

使用select调用函数decode进行简洁条件判断。

```
uxdb=# select decode(2, 2, 'text', 3, 'text1', 4, 'text2', 5, 'text3');
  decode
-----------
 text
(1 rows)
```

11.4. 扩展特性

11.4.1. 函数兼容

11.4.1.1. dateadd

11.4.1.1.1. 概述

- 功能

  该函数为日期计算函数主要是计算，从当前日期开始经过多少日、季、月、年等后的日期。

- 参数说明

```
\df oracle.dateadd
Schema | Name | Result data type | Argument data types | Type
--------+---------+----------------------------+----------------------------------------------------------------------+------
 oracle | dateadd | timestamp without time zone | p_component text, p_number integer, date_text timestamp with time zone | func
(1 row)
```

表 11.11. dateadd 参数说明

名称	描述
p_component	时间元件，可以是：年、月、日、时、分、秒、季度、周，如下所示。 • 年（year、yyyy、y、YEAR、YYYY、Y） • 月（month、mon、mm、m、MONTH、MON、MM、M） • 日（day、dd、d、DAY、DD、D） • 时（hour、hh、h、HOUR、HH、H） • 分（minute、mi、MINUTE、MI） • 秒（second、ss、s、SECOND、SS、S） • 季度（quarter、qq、q、QUARTER、QQ、Q） • 周（week、wk、ww、w、WEEK、WK、WW、W）

名称	描述
p_Number	加数，注意：应该为整数（可正可负）
date_text	基准时间

11.4.1.1.2. 用法示例

分别增加年、月、日、时，如下所示。

```
uxdb=# select oracle.dateadd('year', 1, '2020-12-15 16:51:37.309153+08');
        dateadd
----------------------------
 2021-12-15 16:51:37.309153
 (1 row)
uxdb=# select oracle.dateadd('month', 1, '2020-12-31 16:51:37.309153+08');
        dateadd
----------------------------
 2021-01-31 16:51:37.309153
(1 row)
uxdb=# select oracle.dateadd('day', 1, '2020-12-31 16:51:37.309153+08');
        dateadd
----------------------------
 2021-01-01 16:51:37.309153
 (1 row)
uxdb=# select oracle.dateadd('hour', 1, '2020-12-31 16:51:37.309153+08');
        dateadd
----------------------------
 2020-12-31 17:51:37.309153
 (1 row)
```

11.4.1.2. datediff

11.4.1.2.1. 概述

- 功能

  函数返回两个日期之间相差的天数。

- 参数说明

```
\df oracle.datediff
 Schema |   Name   | Result data type |                       Argument data types                        | Type
--------+----------+------------------+--------------------------------------------------------------------------------
+------
 oracle | datediff | integer          | p_subtranhend timestamp with time zone, p_minuend timestamp with
 time zone | func
(1 row)
```

表 11.12. datediff 参数说明

名称	描述
p_subtrahend	被减数时间
p_minuend	减数时间

11.4.1.2.2. 用法示例

计算出相差的天数，如下所示。

```
uxdb=# select oracle.datediff('2020-12-15 16:51:37.309153+08', '2021-12-15 16:51:37.309153+08');
 datediff
----------
     -365
(1 row)
uxdb=# select oracle.datediff('2020-12-31 16:51:37.309153+08', '2020-11-30 16:51:37.309153+08');
 datediff
----------
       31
(1 row)
uxdb=# select oracle.datediff('2020-3-1 16:51:37.309153+08', '2019-3-1 16:51:37.309153+08');
datediff
----------
      366
(1 row)
uxdb=# select oracle.datediff('2019-3-1 16:51:37.309153+08', '2018-3-1 16:51:37.309153+08');
 datediff
----------
      365
(1 row)
```

11.4.1.3. sysdate

11.4.1.3.1. 概述

sysdate是一个返回数据库服务器所在操作系统，当前日期和时间的关键字。

使用场景

1. 可直接通过命令获取当前日期时间。

   ```
   select sysdate;
   ```

2. 表中含date列，可在where条件中直接使用sysdate进行比较，如下所示。

   ```
   select * from t1 where date > sysdate;
   ```

3. 加减法操作：加/减一天或一星期，加一星期，如下所示。

   ```
   select sysdate,to_char(sysdate+7,'yyyy-mm-dd HH24:MI:SS');
   ```

11.4.1.3.2. 用法示例

1. 打开语法开关，详情请参见《优炫数据库兼容性使用手册 V2.1》“语法兼容”章节中的“语法开关”。

   ```
   set ora_grammar = on;
   ```

2. 执行如下命令并查看返回结果。

   ```
   uxdb=# select sysdate;
   ```

```
      sysdate
---------------------
 2021-07-15 09:30:38
(1 row)
```

3. 创建表t1，包含date列，插入2条及以上数据，date列日期覆盖当前日期前后。

```
uxdb=# create table t1 (id int, date date);
CREATE TABLE
uxdb=# insert into t1 values (1, '2021-07-14'), (2, '2021-07-16');
INSERT 0 2
uxdb=# select * from t1;
 id |    date
----+------------
  1 | 2021-07-14
  2 | 2021-07-16
(2 rows)
```

4. 获取date > sysdate的日期，执行如下命令并查看返回结果。

```
uxdb=# select * from t1 where date > sysdate;
 id |    date
----+------------
  2 | 2021-07-16
(1 row)
```

5. 加法，执行如下命令并查看返回结果。

```
//加1星期
uxdb=# select sysdate,to_char(sysdate+7,'yyyy-mm-dd HH24:MI:SS');
      sysdate        |       to_char
---------------------+---------------------
 2021-07-15 09:36:15 | 2021-07-22 09:36:15
(1 row)
//加1天
uxdb=# select sysdate,to_char(sysdate+1,'yyyy-mm-dd HH24:MI:SS');
      sysdate        |       to_char
---------------------+---------------------
 2021-07-15 09:37:43 | 2021-07-16 09:37:43
(1 row)
```

6. 减法，执行如下命令并查看返回结果。

```
//减1星期
uxdb=# select sysdate,to_char(sysdate-7,'yyyy-mm-dd HH24:MI:SS');
      sysdate        |       to_char
---------------------+---------------------
 2021-07-15 09:38:30 | 2021-07-08 09:38:30
(1 row)
//减1天
uxdb=# select sysdate,to_char(sysdate-1,'yyyy-mm-dd HH24:MI:SS');
      sysdate        |       to_char
---------------------+---------------------
 2021-07-15 09:39:49 | 2021-07-14 09:39:49
(1 row)
```

11.4.1.4. to_char

11.4.1.4.1. 概述

to_char(text,text)是orafce插件中to_char函数的补充，功能是将字符串形式的时间戳转换为指定格式的字符串输出。

参数说明

- 参数1（text类型）

  需要转换的字符串形式的时间戳。

  1. 标准模式下，UXDB对时间戳类型不会自动进行分隔，此参数仅支持带分隔符格式的时间戳字符串，如：'2021-01-02 03:04:05'；
  2. 兼容模式下，UXDB添加了对时间戳类型的自动分隔功能，此参数除支持带分隔符格式的时间戳字符串外，还可以支持无分隔符格式的时间戳字符串，如：'20210102030405'；

- 参数2（text类型）

  用于转换的模板，即所要输出的字符串格式，如：'YYYYMMDDHH24MISS'；或'YYYYMMDDHH12MISS'；

  在模板字符中允许加入一些特殊符号作为分隔符，具体模板匹配使用方法说明和匹配格式的字符格式需参考表 11.13 “to_char函数转换模板”，仅部分有特殊含义的，支持首字母大写。

11.4.1.4.2. 用法示例

执行如下命令，查看返回结果。

```
uxdb=# select to_char('2021-01-02 03:04:05','YYYYMMDDHH24MISS');
   to_char
----------------
 20210102030405
(1 row)
uxdb=# select to_char('2021-01-02 03:04:05','YYYY/MM/DD HH24.MI.SS');
      to_char
---------------------
 2021/01/02 03.04.05
(1 row)
```

11.4.1.4.3. 转换模板

表 11.13. to_char函数转换模板

模板	描述
YYYY	年（4位）
YYY	年的后三位
YY	年的后两位
Y	年的最后一位
CC	世纪（2位）
J	Julian日期（儒略日，自公元前4712年1月1日来的日期）

模板	描述
Q	季度
MONTH	全长大写月份名（9字符）
Month	全长首字母大写月份名（9字符）
month	全长小写月份名（9字符）
MON	大写缩写月份名（3字符）
Mon	缩写首字母大写月份名（3字符）
mon	小写缩写月份名（3字符）
MM	月份(01-12)
DAY	全长大写星期名（9字符）
Day	全长首字母大写星期名（9字符）
day	全长小写星期名（9字符）
DY	缩写大写星期名（3字符）
Dy	缩写首字母大写星期名（3字符）
dy	缩写小写星期名（3字符）
DDD	一年里的第几天(001-366)
DD	一月里的第几天(01-31)
D	一周里的第几天(1-7；SUN=1)
HH	一天的小时数(01-12)
HH12	一天的小时数(01-12)
HH24	一天的小时数(00-23)
MI	分钟(00-59)
SS	秒(00-59)
SSSS	午夜后的秒(0-86399)
RM	罗马数字的月份（I-XII；I=JAN）－大写
rm	罗马数字的月份（I-XII；I=JAN）－小写
AM or A.M. or PM or P.M.	正午标识（大写）
am or a.m. or pm or p.m.	正午标识（小写）
BC or B.C. or AD or A.D.	纪年标识（大写）
bc or b.c. or ad or a.d.	纪年标识（小写）

11.4.1.5. to_date

11.4.1.5.1. 概述

to_date(bigint,text)，函数将给定bigint类型的数据按照yyyy-mm-dd hh-24(12)-miss的表示类型转换成时间类型数据。

在orafce扩展中创建函数to_date函数映射，将bigint类型数据处理转换成时间时间类型数据。

需在兼容模式下且加载orafce扩展，如下所示。

UXDB=# **\df to_date**

```
函数列表
架构模式 | 名称 | 结果数据类型 |参数数据类型|类型
-----------+---------+-------------------------------+---------------+------
UX_CATALOG | TO_DATE | TIMESTAMP WITH TIME ZONE | BIGINT, TEXT | 函数
UX_CATALOG | TO_DATE | TIMESTAMP WITHOUT TIME ZONE | STR TEXT | 函数
UX_CATALOG | TO_DATE | DATE | TEXT, TEXT | 函数
```

以下仅测试Select to_date（bigint，text）实现

11.4.1.5.2. 用法示例

```
UXDB=# SELECT to_date(20210812143356,'yyyymmddhh24miss');
 TO_DATE
------------------------
 2021-08-12 14:33:56+08
(1 行记录)
UXDB=# SELECT to_date(20210812148856,'yyyymmddhh24miss');
错误: 日期/时间值超出范围: "20210812148856"
UXDB=# SELECT to_date(20210812143378,'yyyymmddhh24miss');
错误: 日期/时间值超出范围: "20210812143378"
UXDB=# SELECT to_date(20210812103356,'yyyymmddhh24miss');
 TO_DATE
------------------------
 2021-08-12 10:33:56+08
(1 行记录)
UXDB=# SELECT to_date(202108121433561234,'yyyymmddhh24miss');
错误: 日期/时间值超出范围: "202108121433561234"
UXDB=# SELECT to_date(20210812143356,'yyyymmddhh');
错误: 对于12小时制的钟表，小时数"143356"无效
提示: 使用24小时制的钟表, 或者将小时数限定在1到12之间.
UXDB=# SELECT to_date(20210812083356,'yyyymmddhh12miss');
 TO_DATE
------------------------
 2021-08-12 08:33:56+08
(1 行记录)
UXDB=# SELECT to_date(20210812088856,'yyyymmddhh12miss');
错误: 日期/时间值超出范围: "20210812088856"
UXDB=# SELECT to_date(20210812143344,'yyyymmddhh12miss');
错误: 对于12小时制的钟表，小时数"14"无效
提示: 使用24小时制的钟表, 或者将小时数限定在1到12之间.
UXDB=# SELECT to_date(20210812103399,'yyyymmddhh12miss');
错误: 日期/时间值超出范围: "20210812103399"
```

第 12 章 ux_implicit

12.1. 概述

UXDB内置了100多种隐式转换，由于其基本是同类别之间的隐式转换，所以使用场景会有限制。为了满足更多的使用场景以及更好的使用体验，我们定义了一个新的隐式转换插件：ux_implicit，里面包含了更多的隐式转换的内容。

要想使用该插件，在标准模式下，可以通过命令“create extension ux_implicit;”进行创建，创建成功后即可使用；在兼容模式下，该插件是默认加载的，可以直接使用。

12.2. 隐式转换类型

表 12.1. 字符类型到数字类型

源类型	目标类型
character、varchar、text、name	smallint、integer、bigint、real、dobule precision、numeric
varchar2、nvarchar2	smallint、integer、bigint、real、double precision、numeric

表 12.2. 字符类型到时间类型和时间间隔类型

源类型	目标类型
character、varchar、text、name	timestamp without time zone、timestamp with time zone、time without time zone、time with time zone、interval、date
varchar2、nvarchar2	date、timestamp without time zone、interval

表 12.3. 布尔类型到数字类型

源类型	目标类型
boolean	smallint、integer、bigint、real、dobule precision、numeric

表 12.4. 数字类型到字符类型

源类型	目标类型
smallint、integer、bigint、real、dobule precision、numeric	varchar2、nvarchar2

表 12.5. 时间类型和时间间隔类型到字符类型

源类型	目标类型
date、timestamp without time zone、interval	varchar2、nvarchar2

表 12.6. 字符类型到布尔类型

源类型	目标类型
character、varchar、text、name	boolean

12.3. 支持场景

1. 支持字符类型转换为数值类型的操作符调用和函数调用。

   //t_ch(ch text);
   //abs(ch) => abs(ch::numeric);
   select abs(ch) from t_ch;
   //ch = 1 => ch::integer = 1
   select * from t_ch where ch = 1;

2. 支持字符类型转换为布尔类型的操作符调用和函数调用。

   //t_ch(ch text);
   //ch='t'::boolean => ch::boolean='t'::boolean
   select * from t_ch where ch = 't'::boolean;
   //every(ch) => every(ch::boolean)
   select every(ch) from t_ch;

3. 支持字符类型转换为时间类型的操作符调用和函数调用。

   //t_ch(ch text);
   //ch = now() => ch::timestamptz = now()
   select * from t_ch where ch = now();
   //age(ch)=>age(ch::timestamptz)
   select age(ch) from t_ch;

4. 支持布尔类型转换为数值类型的操作符调用。

   //t_bool(b bool);
   //abs(b) => abs(ch::numeric);
   select abs(b) from t_bool;
   //b=1 => b::integer=1
   select * from t_bool where b = 1;

5. 支持字符类型的表使用btree索引与其他非字符类型的表进行查询。

   例如，对于两张表，其字段分别为字符类型和非字符类型。在字符类型的字段上创建btree索引，在另一张表查询时，字符类型上的btree索引会生效。

   > **注意**
   >
   > 如果数据量小且enable_seqscan状态为on，不会使用索引扫描；如果enable_seqscan状态为off，则会使用索引扫描。

6. 支持非字符类型的表使用btree与hash索引与字符类型的表进行查询（timestamptz、real和bool类型的表不支持使用hash索引与字符类型的表进行查询）。

   例如，对于两张表，其字段分别为非字符类型和字符类型。在非字符类型的字段上创建btree索引，在另一张表查询时，非字符类型上的btree索引会生效；在非字符类型的字段上创建hash索引，在另一张表查询时，除timestamptz、bool和real这三种类型上的hash索引不生效外，其余非字符类型上的hash索引会生效。

7. 支持字符类型的表使用btree与hash索引与字符类型的表进行查询。

例如，对于两张表，其字段都是字符类型。在任意表的字符类型的字段上创建btree索引，在另一张表查询时，字符类型上的btree索引会生效。

8. 不支持字符类型的表使用hash索引与其他非字符类型的表进行查询。

   例如，对于两张表，其字段分别为字符类型和非字符类型。在字符类型的字段上创建hash索引，在另一张表查询时，字符类型上的hash索引不会生效。

> **提示**
>
> 尽管UXDB会自动调用隐式转换，来转换数据类型，但是这种类型转换是固定的，因此使用隐式转换可能会造成错误。例如，有一个表t_ch(ch text)进行如下调用必须使用显示转换：
>
> **select abs(ch::integer) from t_ch;1 + ch::bigint from t_ch;**
>
> 如果使用隐式转换,则会发生如下转换：
>
> abs(ch) => abs(ch::numeric)； 1 + ch => 1::integer + ch::integer

支持隐式转换的函数和隐式转换方向，请参见下表。

表 12.7. 隐式转换的函数及方向

函数	初始类型	转换方向
abs	ch\boolean	numeric
acos	ch\boolean	double precision
acosd		
acosh		
asin		
asind		
asinh		
atan		
atanh		
cbrt		
cos		
cosd		
cosh		
cot		
cotd		
dcbrt		
degrees		
dexp		
dlog1		

函数	初始类型	转换方向
dlog10		
dround		
dsqrt		
dtrunc		
radians		
sin		
sind		
sinh		
tan		
tand		
tanh		
var_pop		
var_samp		
variance		
stddev_pop		
stddev_samp		
stddev		
age	ch	timestamptz
	ch,ch	timestamptz,timestamptz
	ch,timestamptz	
	timestamptz,ch	
	ch,timestamp	timestamp,timestamp
	timestamp,ch	
atan2	ch,ch\ boolean,boolean	double precision, double precision
	ch,double precision\ boolean,double precision	
	double precision,ch\ double precision,boolean	
atan2d	ch,ch\ boolean,boolean	double precision, double precision
	ch,double precision\ boolean,double precision	
	double precision,ch\ double precision,boolean	
ceil	ch\boolean	numeric
ceiling		
date	ch	timestamptz
date_part	ch,ch	text,timestamptz
date_trunc	ch,ch	text,timestamptz
	text,ch,text	text,timestamptz,text

函数	初始类型	转换方向
dpow	ch,ch\ boolean,boolean	double precision, double precision
	ch,double precision\ boolean,double precision	
	double precision,ch\ double precision,boolean	
exp	ch\boolean	numeric
float4		
float8		
floor		
int2		
int4		
int8		
ln		
log10		
numeric_abs		
numeric_sqrt		
scale		
sign		
sqrt		
sum		
avg		
numeric_fac	ch\boolean	bigint
log	ch\boolean	numeric
	ch,ch\ boolean,boolean	numeric,numeric
	numeric,ch\ numeric,boolean	
	ch,numeric\ boolean,numeric	
factorial	ch\boolean	bigint
int2abs		smallint
int4abs		integer
int8abs		bigint
float4abs		real
float8abs		double precision
hash_numeric		numeric
hash_numeric_extended	ch,ch\ boolean,boolean	numeric,bigint
	numeric,ch\ numeric,boolean	
	ch,bigint\ boolean,bigint	

函数	初始类型	转换方向
hashfloat4	ch\boolean	real
hashfloat4extended	ch,ch\ boolean,boolean	real,bigint
	real,ch\ real,boolean	
	ch,bigint\ boolean,bigint	
hashfloat8	ch\boolean	double precision
hashfloat8extended	ch,ch\ boolean,boolean	double precision,bigint
	double precision,ch\ double precision,boolean	
	ch,bigint\ boolean,bigint	
hashint2	ch\boolean	smallint
hashint2extended	ch,ch\ boolean,boolean	smallint,bigint
	smallint,ch\ smallint,boolean	
	ch,bigint\ boolean,bigint	
hashint4	ch\boolean	integer
hashint4extended	ch,ch\ boolean,boolean	integer,bigint
	integer,ch\ integer,boolean	
	ch,bigint\ boolean,bigint	
hashint8	ch\boolean	bigint
hashint8extended	ch,ch\ boolean,boolean	bigint,bigint
	bigint,ch\ bigint,boolean	
	ch,bigint\ boolean,bigint	
interval_hash	ch	interval
interval_hash_extended	ch,ch	interval,bigint
	interval,ch	
	ch,bigint	
time_hash	ch	time
time_hash_extended	ch,ch	time,bigint
	time,ch	
	ch,bigint	
timetz_hash	ch	timetz
timetz_hash_extended	ch,ch	timetz,bigint
	timetz,ch	
	ch,bigint	
timestamp_hash	ch	timestamp
timestamp_hash_extended	ch,ch	timestamp,bigint
	timestamp,ch	
	ch,bigint	

函数	初始类型	转换方向
numeric_power	ch,ch\ boolean,boolean	numeric,numeric
	ch,numeric\ boolean,numeric	
	numeric,ch\ numeric,boolean	
pow	ch,ch\ boolean,boolean	numeric,numeric
	ch,numeric\ boolean,numeric	
	numeric,ch\ numeric,boolean	
power	ch,ch\ boolean,boolean	numeric,numeric
	ch,numeric\ boolean,numeric	
	numeric,ch\ numeric,boolean	
bool_and	ch	boolean
every		
bool_or		
regr_count	• ch,ch\ boolean,boolean • ch,double precision\ boolean,double precision • double precision,ch\ double precision,boolean	double precision, double precision
regr_sxx		
regr_syy		
regr_sxy		
regr_avgx		
regr_avgy		
regr_r2		
regr_slope		
regr_intercept		
corr		
covar_pop		
covar_samp		
div	ch,ch\ boolean,boolean	numeric,numeric
	ch,numeric\ boolean,numeric	
	numeric,ch\ numeric,boolean	
mod	ch,ch\ boolean,boolean	numeric,numeric
	ch,numeric\ boolean,numeric	
	numeric,ch\ numeric,boolean	
bit_and	ch\boolean	bigint

函数	初始类型	转换方向
bit_or		
round	ch\boolean	double precision
trunc		
isfinite	ch	timestamptz
to_char	ch,ch	timestamp,text
	boolean,ch	double precision,text

支持隐式转换的操作符和隐式转换方向，请参见下表。

表 12.8. 隐式转换的操作符及方向

操作符	输入	转换方向
=、>、<、>=、<=、<>	boolean,ch\ ch,boolean	boolean,boolean
	real,ch\ ch,real	real,real
	real,boolean\ boolean,real	
	double precision,ch\ ch,double precision	double precision, double precision
	double precision,boolean\ boolean,double precision	
	smallint,ch\ ch,smallint	smallint,smallint
	smallint,boolean\ boolean,smallint	
	integer,ch\ ch,integer	integer,integer
	integer,boolean\ boolean,integer	
	bigint,ch\ ch,bigint	bigint,bigint
	bigint,boolean\ boolean,bigint	
	numeric,ch\ ch,numeric	numeric,numeric
	numeric,boolean\ boolean,numeric	
	date,ch\ ch,date	date,date
	timestamp,ch\ ch,timestamp	timestamp,timestamp
	timestamptz,ch\ ch,timestamptz	timestamptz,timestamptz
	time,ch\ ch,time	time,time
	timetz,ch\ ch,timetz	timetz,timetz
+、-、*、/	real,ch\ ch,real	real,real
	real,boolean\ boolean,real	
	double precision,ch\ ch,double precision	double precision,double precision

操作符	输入	转换方向
	double precision,boolean\ boolean,double precision	
	smallint,ch\ ch,smallint	smallint,smallint
	smallint,boolean\ boolean,smallint	
	integer,ch\ ch,integer	integer,integer
	integer,boolean\ boolean,integer	
	bigint,ch\ ch,bigint	bigint,bigint
	bigint,boolean\ boolean,bigint	
	numeric,ch\ ch,numeric	numeric,numeric
	numeric,boolean\ boolean,numeric	
+	timestamp,ch	timestamp,numeric
	ch,timestamp	numeric,timestamp
	timestamptz,ch	timestamptz,numeric
	ch,timestamptz	numeric,timestamptz
	date,ch	date,numeric
	ch,date	numeric, date
-	timestamp,ch	timestamp,numeric
	timestamptz,ch	timestamptz,numeric
	date,ch	date,numeric
<<、>>	smallint,ch\ ch,smallint	smallint,integer
	smallint,boolean\ boolean,smallint	
	integer,ch\ ch,integer	integer,integer
	integer,boolean\ boolean,integer	
	bigint,ch\ ch,bigint	bigint,integer
	bigint,boolean\ boolean,bigint	
%	smallint,ch\ ch,smallint	smallint,smallint
	smallint,boolean\ boolean,smallint	
	integer,ch\ ch,integer	integer,integer
	integer,boolean\ boolean,integer	
	bigint,ch\ ch,bigint	bigint,bigint
	bigint,boolean\ boolean,bigint	

操作符	输入	转换方向
	numeric,ch\ ch,numeric	numeric,numeric
	numeric,boolean\ boolean,numeric	
^	double precision,ch\ ch,double precision	double precision, double precision
	double precision,boolean\ boolean,double precision	
	numeric,ch\ ch,numeric	numeric,numeric
	numeric,boolean\ boolean,numeric	
!、!!	ch	bigint
	boolean	
|/、||/	ch	double precision
	boolean	
@	ch	numeric
	boolean	
&、|、#	smallint,ch\ ch,smallint	smallint,smallint
	smallint,boolean\ boolean,smallint	
	integer,ch\ ch,integer	integer,integer
	integer,boolean\ boolean,integer	
	bigint,ch\ ch,bigint	bigint,bigint
	bigint,boolean\ boolean,bigint	
~	ch	bigint
	boolean	
+、-	ch	numeric
	boolean	
*	interval,ch\ interval,boolean	interval,double precision
	ch,interval\ boolean,interval	double precision,interval
备注：字符类型用ch表示		

12.4. 示例

1. insert\update的隐式转换。

create table t1(c1 int);
create table t2(c2 text);
insert into t1 select * from t2;

2. 调用函数的隐式转换。

```
create table t1(c1 text);
select abs(c1) from t1;
```

3. 调用操作符的隐式转换。

```
create table t1(c1 text);
select * from t1 where c1 = 1;
```

第 13 章 postgres_adaptor

UXDB提供了postgres_adaptor插件，使第三方工具能够正常访问UXDB，达到良好的功能与性能效果。通过适配与UXDB的交互方式，使以SQL语句为手段提供专用功能的第三方工具与UXDB完全兼容。

通过postgres_adaptor使第三方工具与UXDB适配，允许第三方工具使用其自身的变量、函数命名与函数调用方式。将第三方工具的存储过程存储在数据库系统中，并在系统信息中进行注册，使得在调用的过程中能够获取并执行相应过程。

postgres_adaptor分为两部分，输入的适配与输出的适配。

- 输入适配指的是第三方工具的输入能使UXDB识别并正确执行。
- 输出适配指的是第三方工具需要读取UXDB的系统信息并进行判断与验证时，能识别UXDB输出的数据格式。

表 13.1. 输入适配与输出适配

适配类型	项目	描述
输入适配	配置文件	uxsinodb.conf/shared_preload_libraries参数。
	功能描述	输入的SQL语句、存储过程和函数等，转换为postgres兼容的格式。
	参数说明	shared_preload_libraries 参数选项中包含'postgres_adaptor'时，本功能开启；否则本功能关闭。修改后需要重启UXDB服务器。
输出适配	配置文件	uxsinodb.conf/adaptor_pg_ux_mode_switch参数。
	功能描述	SQL语句、存储过程和函数等输出的执行结果，转换为postgres兼容的格式。
	参数说明	在输入适配开启的前提下： adaptor_pg_ux_mode_switch = on时，本功能开启；否则本功能关闭。修改后需要重启UXDB服务器。

UXDB使用ux_*作为内部系统的命名标识，所有系统函数、系统信息以ux_开头。为了使UXDB兼容第三方工具，在adaptor内完成数据库对象从pg_*到ux_*的重命名，以及适配一系列对应的存储过程和函数的调用过程。

数据库对象名称pg_* -> ux_*之间的转换是一对一的转换表， 采用文本文件的方式进行配置。

- 文件为UXDB安装目录下的dbsql/share/extension/postgres_adaptor.data，可在版本发布后根据需要增加或者删除转换项目。如果转换列表文件有变化，需要重启DB server服务使其生效。

- 每个配置项的最大长度为63个字符，最大支持2048个转换项目。
- 数据格式为：

  pg_catalog = ux_catalog

  pg_class = ux_class

- 这些配置项需要保证字典序排序。
- 以#开始的行为注释行。

第 14 章 tablefunc

tablefunc模块包括各种返回表的函数。这些函数都很有用，并且也可以作为如何编写返回多行的C函数的示例。

表 14.1. tablefunc函数

函数	返回	描述
normal_rand(int numvals, float8 mean, float8 stddev)	setof float8	产生一个正态分布的随机值集合。
crosstab(text sql)	setof record	产生一个包含行名称外加N个值列的“数据透视表”，其中N由调用查询中指定的行类型决定。
crosstabN(text sql)	setof table_crosstab_N	产生一个包含行名称外加N个值列的“数据透视表”。crosstab2、crosstab3和crosstab4是被预定义的，但可以按照下文所述创建额外的crosstabN函数。
crosstab(text source_sql, text category_sql)	setof record	产生一个“数据透视表”，其值列由第二个查询指定。
connectby(text relname, text keyid_fld, text parent_keyid_fld [, text orderby_fld], text start_with, int max_depth [, text branch_delim])	setof record	生成分层树结构的表示形式。

使用tablefunc模块前，首先需要执行CREATE EXTENSION命令。

CREATE EXTENSION tablefunc;

14.1. normal_rand

normal_rand(int numvals, float8 mean, float8 stddev) returns setof float8

normal_rand：产生一个正态分布随机值（高斯分布）的集合。

numvals：是从该函数返回的值的数量，mean是值的正态分布的均值，而stddev是值的正态分布的标准偏差。

例如，这个调用请求1000个值，它们具有均值5和标准偏差3。

```
SELECT * FROM normal_rand(1000, 5, 3);
     normal_rand
----------------------
     1.56556322244898
     9.10040991424657
     5.36957140345079
```

```
 -0.369151492880995
  0.283600703686639
   .
   .
   .
  4.82992125404908
  9.71308014517282
  2.49639286969028
(1000 rows)
```

14.2. crosstab(text)

crosstab(text sql)

crosstab函数被用来产生“pivot”显示，在其中数据被横布在页面上而不是直接向下列举。可能存在如下所示的数据。

```
row1    val11
row1    val12
row1    val13
...
row2    val21
row2    val22
row2    val23
...
```

而希望显示成这样：

```
row1    val11   val12   val13   ...
row2    val21   val22   val23   ...
...
```

crosstab函数接受文本参数，该参数是一个SQL查询，生成以第一种方式格式化的原始数据和以第二种方式格式化的表。

sql参数是一个生成源数据集的SQL语句。这个语句必须返回一个row_name列、一个category列和一个value列。

例如，所提供的查询是这样的一个集合：

```
 row_name   cat   value
----------+-------+-------
 row1     cat1   val1
 row1     cat2   val2
 row1     cat3   val3
 row1     cat4   val4
 row2     cat1   val5
 row2     cat2   val6
 row2     cat3   val7
 row2     cat4   val8
```

crosstab函数被声明为返回setof record， 因此输出列的实际名称和类型必须定义在调用的SELECT语句的FROM子句中，如下所示。

SELECT * FROM crosstab('...') AS ct(row_name text, category_1 text, category_2 text);

这个示例产生这样一个集合。

```
  <== value  columns  ==>
 row_name   category_1   category_2
----------+------------+------------
 row1       val1         val2
 row2       val5         val6
```

FROM子句必须把输出定义为一个row_name列（具有SQL查询的第一个结果列的相同数据类型），其后跟随着N个value列（都具有SQL查询的第三个结果列的相同数据类型）。可以按照意愿设置任意多的输出值列。而输出列的名称取决于自己。

crosstab函数为具有相同row_name值的输入行的每一个连续分组产生一个输出行。它使用来自这些行的值域从左至右填充输出的值列。如果一个分组中的行比输出值列少，多余的输出列将被用空值填充。如果行更多，则多余的输入行会被跳过。

事实上，SQL查询应该总是指定ORDER BY 1,2来保证输入行被正确地排序，也就是说具有相同row_name的值会被放在一起并且在行内被正确地排序。注意crosstab本身并不关注查询结果的第二列，它放在那里只是为了被排序，以便控制出现在页面上的第三列值的顺序。

示例

```
CREATE TABLE ct(id SERIAL, rowid TEXT, attribute TEXT, value TEXT);
INSERT INTO ct(rowid, attribute, value) VALUES('test1','att1','val1');
INSERT INTO ct(rowid, attribute, value);
VALUES('test1','att2','val2');
INSERT INTO ct(rowid, attribute, value) VALUES('test1','att3','val3');
INSERT INTO ct(rowid, attribute, value) VALUES('test1','att4','val4');
INSERT INTO ct(rowid, attribute, value) VALUES('test2','att1','val5');
INSERT INTO ct(rowid, attribute, value) VALUES('test2','att2','val6');
INSERT INTO ct(rowid, attribute, value) VALUES('test2','att3','val7');
INSERT INTO ct(rowid, attribute, value) VALUES('test2','att4','val8');
SELECT *
  FROM crosstab(
     'select rowid, attribute, value
        from ct
       where attribute = ''att2'' or attribute = ''att3''
       order by 1,2')
   AS ct(row_name text, category_1 text, category_2 text, category_3 text);
 row_name | category_1 | category_2 | category_3
----------+------------+------------+------------
 test1    | val2       | val3       |
 test2    | val6       | val7       |
(2 rows)
```

为了避免总是要写出一个FROM子句来定义输出列，有以下两种方法。

1. 设置一个在其定义中硬编码所期望的输出行类型的自定义crosstab函数，请参见第 14.3 节“crosstabN(text)”。

2. 在视图定义中嵌入所需的FROM子句。

> **注意**
>
> uxsql中的\crosstabview命令也提供了和crosstab()类似的功能。

14.3. crosstabN(text)

crosstabN(text sql)

crosstabN系列函数是为普通crosstab函数设置自定义封装器的示例，这样不需要在调用的SELECT查询中写出列名和类型。tablefunc模块包括crosstab2、crosstab3以及crosstab4，它们的输入行类型定义如下所示。

```
CREATE TYPE tablefunc_crosstab_N AS (
   row_name TEXT,
   category_1 TEXT,
   category_2 TEXT,
      .
      .
      .
   category_N TEXT
);
```

因此，当输入查询产生类型为text的列row_name和value并且想要2、3或4个输出值列时，函数可以被直接使用。在所有其他方法中，它们的行为都和上面的一般crosstab函数完全相同。

示例

```
SELECT *
  FROM crosstab3(
      'select rowid, attribute, value
        from ct
       where attribute = ''att2'' or attribute = ''att3''
       order by 1,2');
```

这些函数主要是出于举例的目的而提供。可以基于底层的crosstab()函数创建自己的返回类型和函数。有两种方法，如下所示。

- 与uxdbinstall/dbsql/share/extension/tablefunc--1.0.sql中相似，创建一个组合类型来描述所期望的输出列。然后定义一个唯一的函数名，它接受一个text参数并且返回setof your_type_name，但是链接到同样的底层crosstab C函数。例如，如果源数据产生为text类型的行名称，值是float8，并且想要5个值列：

  ```
  CREATE TYPE my_crosstab_float8_5_cols AS (
     my_row_name text,
     my_category_1 float8,
     my_category_2 float8,
     my_category_3 float8,
     my_category_4 float8,
     my_category_5 float8
  );
  CREATE OR REPLACE FUNCTION crosstab_float8_5_cols(text)
  RETURNS setof my_crosstab_float8_5_cols
  ```

```
    AS '$libdir/tablefunc','crosstab' LANGUAGE C STABLE STRICT;
```

• 使用OUT参数来隐式定义返回类型。也可以进行如下操作。

```
CREATE OR REPLACE FUNCTION crosstab_float8_5_cols(
    IN text,
    OUT my_row_name text,
    OUT my_category_1 float8,
    OUT my_category_2 float8,
    OUT my_category_3 float8,
    OUT my_category_4 float8,
    OUT my_category_5 float8)
  RETURNS setof record
    AS '$libdir/tablefunc','crosstab' LANGUAGE C STABLE STRICT;
```

14.4. crosstab(text, text)

crosstab(text source_sql, text category_sql)

crosstab的单一参数形式的主要限制是它把一个组中的所有值都视作相似，并且把每一个值插入到第一个可用的列中。如果想要值列对应于特定的数据分类，或者因为某些分组没有关于某些分类的数据导致无法正常工作。crosstab的双参数形式通过提供一个对应于输出列的显式分类列表来处理这种情况。

source_sql是一个产生源数据集的SQL语句。这个语句必须返回一个row_name列、一个category列以及一个value列。也可以有一个或者多个“extra”列。row_name列必须是第一个。category和value列必须是按照这个顺序的最后两个列。row_name和category之间的任何列都被视作“extra”。对于具有相同row_name值的所有行，其“extra”列都应该相同。

source_sql可能会产生如下内容。

```
SELECT row_name, extra_col, cat, value FROM foo ORDER BY 1;
 row_name    extra_col   cat    value
----------+------------+-----+---------
  row1        extra1    cat1    val1
  row1        extra1    cat2    val2
  row1        extra1    cat4    val4
  row2        extra2    cat1    val5
  row2        extra2    cat2    val6
  row2        extra2    cat3    val7
  row2        extra2    cat4    val8
```

category_sql是一个产生分类集合的SQL语句。这个语句必须只返回一列。它必须返回至少一行，如果是多行则不能重复，否则会生成错误。category_sql示例如下所示。

```
SELECT DISTINCT cat FROM foo ORDER BY 1;
    cat
  -------
    cat1
    cat2
    cat3
    cat4
```

crosstab函数被声明为返回setof record，输出列的实际名称和类型必须在调用的SELECT语句的FROM子句中被定义，如下所示。

```
SELECT * FROM crosstab('...', '...')
  AS ct(row_name text, extra text, cat1 text, cat2 text, cat3 text, cat4 text);
```

这将产生这样的结果：

```
          <== value  columns  ==>
row_name   extra   cat1   cat2   cat3   cat4
---------+-------+------+------+------+------
 row1      extra1  val1   val2          val4
 row2      extra2  val5   val6   val7   val8
```

FROM子句必须定义正确数量的输出列以及正确的数据类型。如果在source_sql查询的结果中有N列，其中的前N-2列必须匹配前N-2个输出列。剩余的输出列必须具有source_sql查询结果的最后一列的类型，并且并且它们的数量必须正好和category_sql查询结果中的行数相同。

crosstab函数为具有相同row_name值的输入行形成的每一个连续分组产生一个输出行。输出的row_name列外加任意一个“extra”列都是从分组的第一行复制而来。输出的value列被使用具有匹配的category值的行中的value域填充。如果一个行的category不匹配category_sql查询的任何输出，它的value会被忽略。匹配的分类不出现于分组中任何输出行中的的输出列会被用空值填充。

事实上，source_sql查询应该总是指定ORDER BY 1来保证具有相同row_name的值会被放在一起。但是，一个分组内分类的顺序并不重要。还有，确保category_sql查询的输出的顺序与指定的输出列顺序匹配是非常重要的。

这里有两个完整的示例。

- 示例一

  ```
  create table sales(year int, month int, qty int);
  insert into sales values(2007, 1, 1000);
  insert into sales values(2007, 2, 1500);
  insert into sales values(2007, 7, 500);
  insert into sales values(2007, 11, 1500);
  insert into sales values(2007, 12, 2000);
  insert into sales values(2008, 1, 1000);
  select * from crosstab(
    'select year, month, qty from sales order by 1',
    'select m from generate_series(1,12) m'
  ) as (
    year int,
    "Jan" int,
    "Feb" int,
    "Mar" int,
    "Apr" int,
    "May" int,
    "Jun" int,
    "Jul" int,
    "Aug" int,
    "Sep" int,
    "Oct" int,
    "Nov" int,
    "Dec" int
  ```

```
);
 year | Jan  | Feb  | Mar | Apr | May | Jun | Jul | Aug | Sep | Oct | Nov  | Dec
------+------+------+-----+-----+-----+-----+-----+-----+-----+-----+------+------
 2007 | 1000 | 1500 |     |     |     | 500 |     |     |     | 1500 | 2000
 2008 | 1000 |      |     |     |     |     |     |     |     |     |
(2 rows)
```

- 示例二

```
CREATE TABLE cth(rowid text, rowdt timestamp, attribute text, val text);
INSERT INTO cth VALUES('test1','01 March 2003','temperature','42');
INSERT INTO cth VALUES('test1','01 March 2003','test_result','PASS');
INSERT INTO cth VALUES('test1','01 March 2003','volts','2.6987');
INSERT INTO cth VALUES('test2','02 March 2003','temperature','53');
INSERT INTO cth VALUES('test2','02 March 2003','test_result','FAIL');
INSERT INTO cth VALUES('test2','02 March 2003','test_startdate','01 March 2003');
INSERT INTO cth VALUES('test2','02 March 2003','volts','3.1234');
SELECT * FROM crosstab
(
  'SELECT rowid, rowdt, attribute, val FROM cth ORDER BY 1',
  'SELECT DISTINCT attribute FROM cth ORDER BY 1'
)
AS
(
       rowid text,
       rowdt timestamp,
       temperature int4,
       test_result text,
       test_startdate timestamp,
       volts float8
);
 rowid |          rowdt           | temperature | test_result |      test_startdate      | volts
-------+--------------------------+-------------+-------------+--------------------------+--------
 test1 | Sat Mar 01 00:00:00 2003 |          42 | PASS        |                          | 2.6987
 test2 | Sun Mar 02 00:00:00 2003 |          53 | FAIL        | Sat Mar 01 00:00:00 2003 | 3.1234
(2 rows)
```

创建预定义的函数可以避免在每个查询中都必须写出结果列的名称和类型，请参见第 14.3 节“crosstabN(text)”中的示例。用于这种形式的crosstab的底层C函数被命名为crosstab_hash。

14.5. connectby

connectby(text relname, text keyid_fld, text parent_keyid_fld [, text orderby_fld], text start_with, int max_depth [, text branch_delim])

connectby函数产生存储在一个表中的分级数据的显示。该表必须具有用以唯一标识行的键字段，以及每一行引用的父键字段（如果有）。connectby能显示从任意行开始向下的子树。

表 14.2. connectby参数

参数	描述
relname	源关系的名称。

参数	描述
keyid_fld	键的名称。
parent_keyid_fld	父键的名称。
orderby_fld	用于排序同级的键名称（可选）。
start_with	起始行的键值。
max_depth	要向下的最大深度，零表示无限深度。
branch_delim	在分支输出中用于分隔键值的字符串（可选）。

键和父键可以是任意数据类型，但是必须是同一类型。注意start_with值必须作为一个文本串被输入，而不管键域的类型如何。

connectby函数被声明为返回setof record，因此输出列的实际名称和类型就必须在调用的SELECT语句的FROM子句中被定义，如下所示。

```
SELECT * FROM connectby('connectby_tree', 'keyid', 'parent_keyid', 'pos', 'row2', 0, '~')
  AS t(keyid text, parent_keyid text, level int, branch text, pos int);
```

前两个输出列是被用于当前行的键和其父行的键，它们必须匹配该表的键的类型。第三个输出列是该树中的深度，并且必须是类型integer。如果给定了一个branch_delim参数，下一个输出列就是分支显示并且必须是类型text。最后，如果给出了一个orderby_fld参数，最后一个输出列是一个序号，并且必须是类型integer。

“branch”输出列显示了用于到达当前行的由键构成的路径。其中的键用指定的branch_delim字符串分隔开。如果不需要分支显示，可以在输出列列表中忽略branch_delim参数和分支列。

如果同一父键域的子键域之间的顺序很重要，可以使用orderby_fld参数指定用域对兄弟键域进行排序。这个域可以是任何可排序数据类型。当且仅当orderby_fld被指定时，输出列列表必须包括一个最终的整数序号列。

表示表和列名的参数会被原样复制到connectby内部生成的SQL查询中。因此，如果名称是大小写混合或者包含特殊字符，应包括双引号。表名需要用模式限定。

在大型的表中，在父键域上需要有索引，否则性能会很差。

branch_delim字符串不能出现在任何键值中，否则connectby可能会错误地报告一个无限递归错误。注意如果没有提供branch_delim，将用默认值（~）来进行递归检测。

示例

```
CREATE TABLE connectby_tree(keyid text, parent_keyid text, pos int);
INSERT INTO connectby_tree VALUES('row1',NULL, 0);
INSERT INTO connectby_tree VALUES('row2','row1', 0);
INSERT INTO connectby_tree VALUES('row3','row1', 0);
INSERT INTO connectby_tree VALUES('row4','row2', 1);
INSERT INTO connectby_tree VALUES('row5','row2', 0);
INSERT INTO connectby_tree VALUES('row6','row4', 0);
INSERT INTO connectby_tree VALUES('row7','row3', 0);
INSERT INTO connectby_tree VALUES('row8','row6', 0);
INSERT INTO connectby_tree VALUES('row9','row5', 0);
```

- 带有分支，但没有orderby_fld（不保证结果的顺序）。

```
SELECT * FROM connectby('connectby_tree', 'keyid', 'parent_keyid', 'row2', 0, '~') AS t(keyid
 text, parent_keyid text, level int, branch text);
 keyid | parent_keyid | level |       branch
-------+--------------+-------+---------------------
 row2  |              |     0 | row2
 row4  | row2         |     1 | row2~row4
 row6  | row4         |     2 | row2~row4~row6
 row8  | row6         |     3 | row2~row4~row6~row8
 row5  | row2         |     1 | row2~row5
 row9  | row5         |     2 | row2~row5~row9
(6 rows)
```

- 没有分支，也没有orderby_fld（不保证结果的顺序）。

```
SELECT * FROM connectby('connectby_tree', 'keyid', 'parent_keyid', 'row2', 0) AS t(keyid
 text, parent_keyid text, level int);
 keyid | parent_keyid | level
-------+--------------+-------
 row2  |              |     0
 row4  | row2         |     1
 row6  | row4         |     2
 row8  | row6         |     3
 row5  | row2         |     1
 row9  | row5         |     2
(6 rows)
```

- 有分支，有orderby_fld，注意row5在row4前面。

```
SELECT * FROM connectby('connectby_tree', 'keyid', 'parent_keyid', 'pos', 'row2', 0, '~') AS
 t(keyid text, parent_keyid text, level int, branch text, pos int);
 keyid | parent_keyid | level |       branch        | pos
-------+--------------+-------+---------------------+-----
 row2  |              |     0 | row2                |   1
 row5  | row2         |     1 | row2~row5           |   2
 row9  | row5         |     2 | row2~row5~row9      |   3
 row4  | row2         |     1 | row2~row4           |   4
 row6  | row4         |     2 | row2~row4~row6      |   5
 row8  | row6         |     3 | row2~row4~row6~row8 |   6
(6 rows)
```

- 没有分支，有orderby_fld，注意row5在row4前面。

```
SELECT * FROM connectby('connectby_tree', 'keyid', 'parent_keyid', 'pos', 'row2', 0) AS
 t(keyid text, parent_keyid text, level int, pos int);
 keyid | parent_keyid | level | pos
-------+--------------+-------+-----
 row2  |              |     0 |   1
 row5  | row2         |     1 |   2
 row9  | row5         |     2 |   3
 row4  | row2         |     1 |   4
 row6  | row4         |     2 |   5
 row8  | row6         |     3 |   6
(6 rows)
```

第 15 章 tds_fdw

15.1. 概述

uxdb具有插件功能，通过不同的插件拓展实现数据库本身不包含的功能，以满足用户的需求。tds_fdw 是一个外部表功能，所谓外部表，就是在UXDB数据库中通过SQL访问外部数据源数据，就像访问本地数据库一样；tds_fdw使用统一的接口方式实现多种数据库的远程访问，包括但不限于sqlserver、sybase等等。

tds_fdw在设置远程访问之后，就可以对映射表进行操作，tds_fdw只支持对外部表的查询操作，即select语句。

15.2. 依赖工具

使用tds_fdw扩展，需要安装freetds工具，使用如下命令进行安装。

```
sudo yum install epel-release
sudo yum install freetds-devel
```

15.3. 示例

1. 安装tds_fdw扩展。

   ```
   uxdb=# create extension tds_fdw;
   CREATE EXTENSION
   ```

2. 使用CREATE SERVER创建一个外部服务器。

   示例中连接的sqlserver服务器主机IP：127.0.0.1；端口：1433；数据库名称：tds_fdw_test，如下所示。

   ```
   uxdb=# CREATE SERVER sybServ FOREIGN DATA WRAPPER tds_fdw OPTIONS
    (servername '127.0.0.1', port '1433', database 'tds_fdw_test');
   CREATE SERVER
   ```

3. 用CREATE USER MAPPING定义一个用户映射来标识在远程服务器上使用哪个角色。

   示例中将服务器定义映射到uxdb用户上，username为sqlserver的用户名，password为sqlserver用户对应的密码，如下所示。

   ```
   uxdb=# CREATE USER MAPPING FOR uxdb SERVER sybServ  OPTIONS (username 'sa',
    password 'tkb123456TKB');
   CREATE USER MAPPING
   ```

4. 使用CREATE FOREIGN TABLE创建外部表。

   示例中table_name为需要映射的表名，如下所示。

   ```
   uxdb=# CREATE FOREIGN TABLE syb_table (id int,name varchar(100))SERVER sybServ
    OPTIONS (table_name 'testtb', row_estimate_method 'showplan_all');
   CREATE FOREIGN TABLE
   ```

5. 从外部映射表中查询数据。

```
uxdb=# select * from syb_table;
 id |   name
----+----------
 10 | xiaowang
(1 row)
```

第 16 章 uxAgent

16.1. 安装依赖软件

1. 安装cmake

   官网下载最新cmake：https://cmake.org/download/。

   - Windows：下载对应的包（后缀名为.msi），直接安装即可。
   - Linux：下载cmake源码，按照如下命令安装。

   ```
   tar -xvf cmake-3.12.3.tar.gz
   cd cmake-3.12.3
   ./bootstrap
   gmake && gmake install
   ```

2. 安装Boost库

   官网下载Boost库，推荐下载1.64.0版本：https://sourceforge.net/projects/boost/files/boost/1.64.0/。

   - Windows：在cmd窗口运行如下命令。

   ```
   bootstrap.bat
   b2.exe link=static address-model=64
   ```

   - Linux：执行如下命令。

   ```
   ./bootstrap.sh
   ./b2 link=static address-model=64
   ```

3. 安装Python

   安装2.7.15版本的Python，并将bin目录加入到系统环境变量PATH中，将lib目录加入到系统环境变量LD_LIBRARY_PATH中。

16.2. uxAgent的安装

16.2.1. Linux下uxAgent的安装

1. 安装uxAgent

   a. 解压安装包uxAgent-4.0.0-Linux.tar.gz。

   ```
   [uxdb@mastermini uxAgent_linux]$ ls
   bin  extension  install.sh  LICENSE  README  share  start.sh  stop.sh
   ```

   b. 执行install.sh进行安装，默认安装路径是/home/uxdb/uxAgent，uxagent可执行路径是/home/uxdb/uxAgent/bin。

   c. 手动配置环境变量。修改内容如下图所示。

```
# Add RVM to PATH for scripting. Make sure this is the last PATH variable change.
export PATH="$PATH:$HOME/.rvm/src/rvm/bin"
# Add sonar-scanner
PATH+=:/home/uxdb/tools/sonar-scanner-3.4.0.1729-linux/bin
UXDBHOME=/home/uxdb/uxdbinstall/dbsql
UX_AGENT_HOME=/home/uxdb/uxdbinstall/uxAgent
# THIRDPARTY_ROOT
THIRDPARTY_ROOT=/home/uxdb/.jenkins/workspace/ref/thirdparty

PATH=$UXDBHOME/bin:$UX_AGENT_HOME/bin:$PATH
export PATH
```

执行如下命令，使配置生效：

source .bashrc

2. 创建uxAgent扩展

初始化数据库实例并启动，使用uxdbAdmin或终端连接至uxdb数据库，创建扩展uxagent。

create extension uxagent;

```
uxdb=# create extension uxagent;
CREATE EXTENSION
uxdb=#
```

3. 启动uxAgent

- 方法一

```
[root@localhost build]# ./uxagent
UXDB Scheduling Agent
Version: 4.0.0
Usage:
./uxagent [options] <connect-string>
options:
-v (display version info and then exit)
-f run in the foreground (do not detach from the terminal)
-t <poll time interval in seconds (default 10)>
-r <retry period after connection abort in seconds (>=10, default 30)>
-s <log file (messages are logged to STDOUT if not specified>
-l <logging verbosity (ERROR=0, WARNING=1, DEBUG=2, default 0)>
[root@localhost build]#
```

uxagent [options] <connect-string>

options（可选参数）

-v

查看版本号信息

-f

在前台运行（默认会启动到后台）

-t

轮询时间间隔，单位是秒，默认值是10

-r

连接中止后的重试时间，单位是秒，一般需要大于能等于10，默认值是30

-s

日志文件

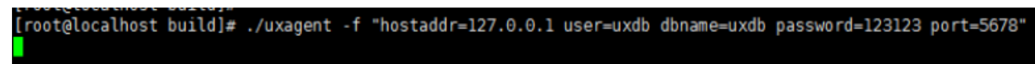
-l

记录详细信息，默认值为0，0表示错误，1表示警告，2表示调试

connect-string表示连接字符串。

启动示例

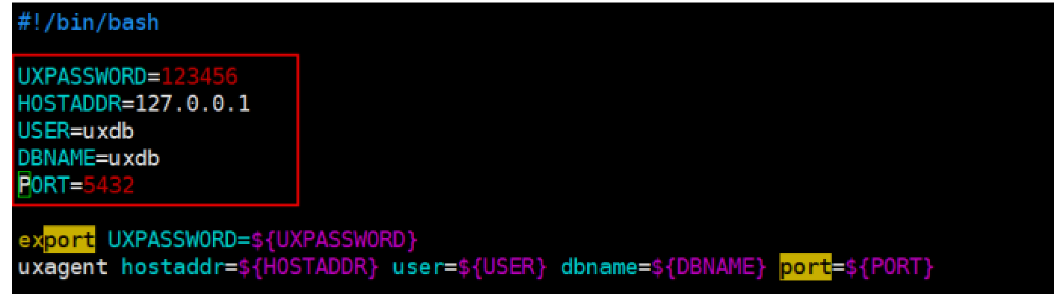
./uxagent -f "hostaddr=127.0.0.1 user=uxdb dbname=uxdb password=123123 port=5678"



> **提示**
>
> 由于完全使用connect-string会暴露密码，因此在启动uxagent之前先将密码赋值给UXPASSWORD环境变量，在启动时去掉password即可。

- 方法二

直接运行start.sh进行启动，注意改脚本中默认数据库为uxdb，端口号为5432，可根据实际情况修改启动脚本。



16.2.2. Windows下uxAgent的安装

1. 安装uxAgent

a. 解压uxAgent_win-4.0.0.zip。

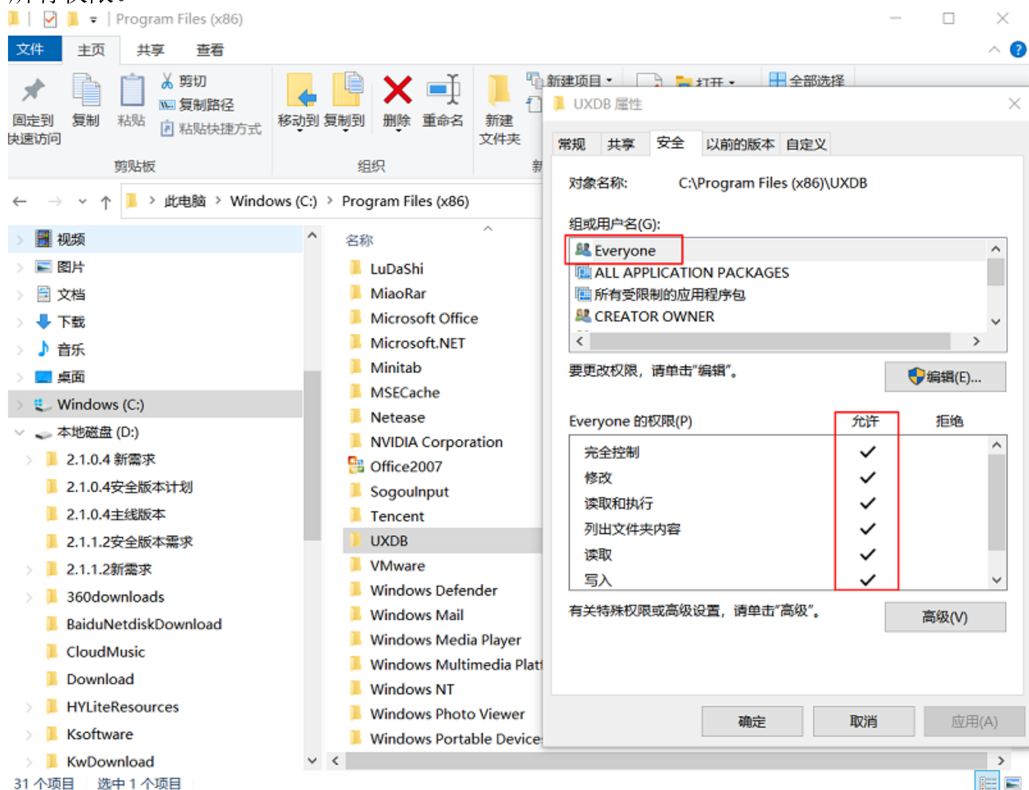
例如：解压路径为E:\UXDB\uxAgent_win-4.0.0。

b. 修改解压目录下的intall.bat中uxdb的安装路径以及uxAgent_win-4.0.0所在的路径（根据具体环境修改）。

```
C:\Windows\System32\cmd.exe - vim install.bat
@echo off

REM Set uxdb installation path
set uxdbinstall_path=E:\UXDB\dbsql      uxdb的安装路径

REM Set the path of uxagent_win-4.0.0 directory after decompression
set uxAgent_path=E:\UXDB\uxAgent_win-4.0.0      解压缩后uxAgent_win-4.0.0目录所在的路径

cd %uxAgent_path%
copy uxagent.exe %uxdbinstall_path%\bin
copy uxagent.control %uxdbinstall_path%\share\extension
copy uxagent--3.4--4.0.sql %uxdbinstall_path%\share\extension
copy uxagent--4.0.sql %uxdbinstall_path%\share\extension
copy uxagent.sql %uxdbinstall_path%\share\extension
copy uxagent--unpackaged--4.0.sql %uxdbinstall_path%\share\extension

cd %uxAgent_path%\boost_lib
xcopy .\*.* %uxdbinstall_path%\lib
pause
```

c. 运行install.bat进行uxagent的安装，安装之后的可执行程序uxagent.exe在uxdb安装目录下的dbsql\bin目录中。

如果uxdb是默认路径安装，即C:\Program Files (x86)\UXDB\dbsql，则需要给UXDB目录所有权限。



2. 创建uxAgent扩展

使用uxdbAdmin或终端连接至uxdb数据库，创建扩展uxagent。

create extension uxagent;

```
uxdb=# create extension uxagent;
CREATE EXTENSION
uxdb=#
```

3. 启动uxAgent

uxagent参数信息可以参见第 16.2.1 节 “Linux下uxAgent的安装”

windows平台需要创建服务，以管理员身份打开创建服务。

uxagent.exe INSTALL uxAgent "hostaddr=192.168.0.165 user=uxdb dbname=uxdb password=123456 port=5432"

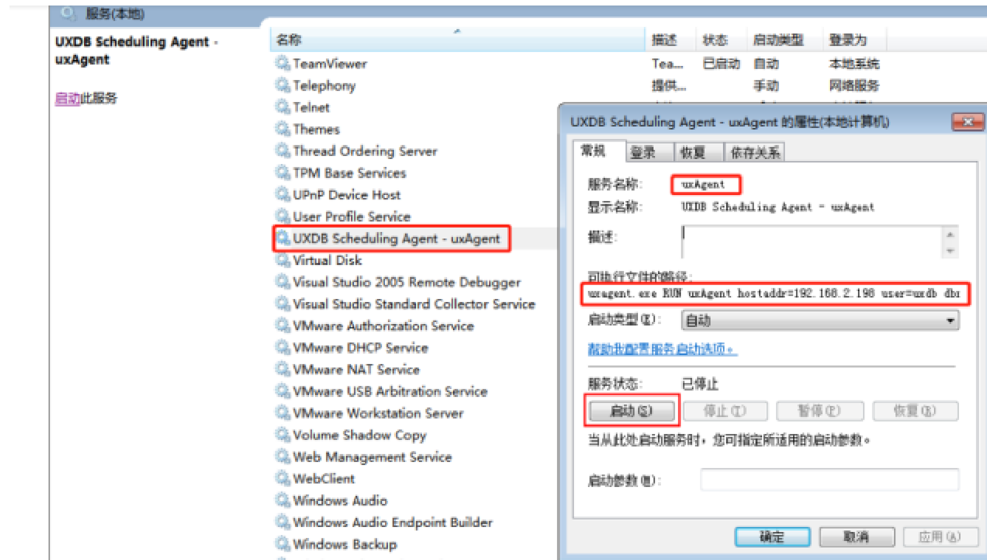
```
E:\UXDB\dbsql\bin>uxagent.exe INSTALL uxAgent "hostaddr=192.168.0.165 user=uxdb
 dbname=uxdb password=123456 port=5432"
```

服务创建后，可将服务的启动用户改为本地系统账户。

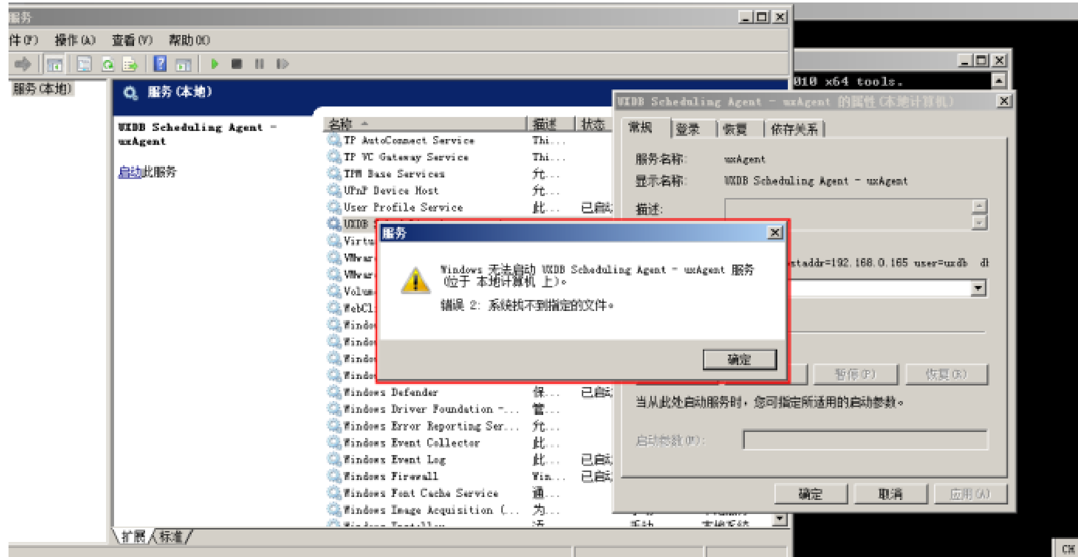
sc config uxAgent obj= LocalSystem

```
E:\UXDB\dbsql\bin>sc config uxAgent obj= LocalSystem
[SC] ChangeServiceConfig 成功
```

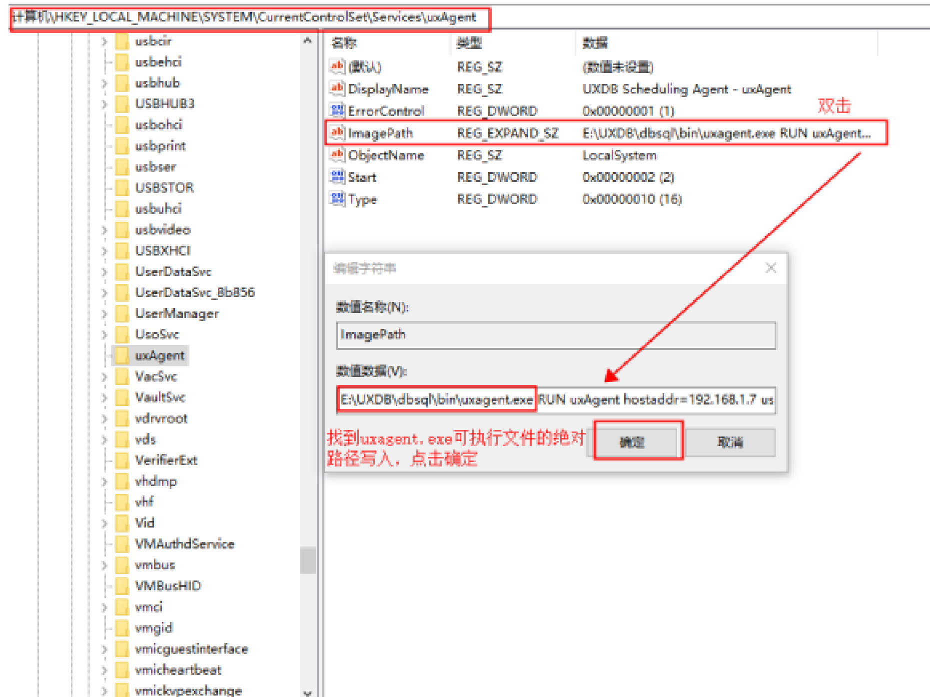
查看服务属性，如下图所示，启动即可使用。



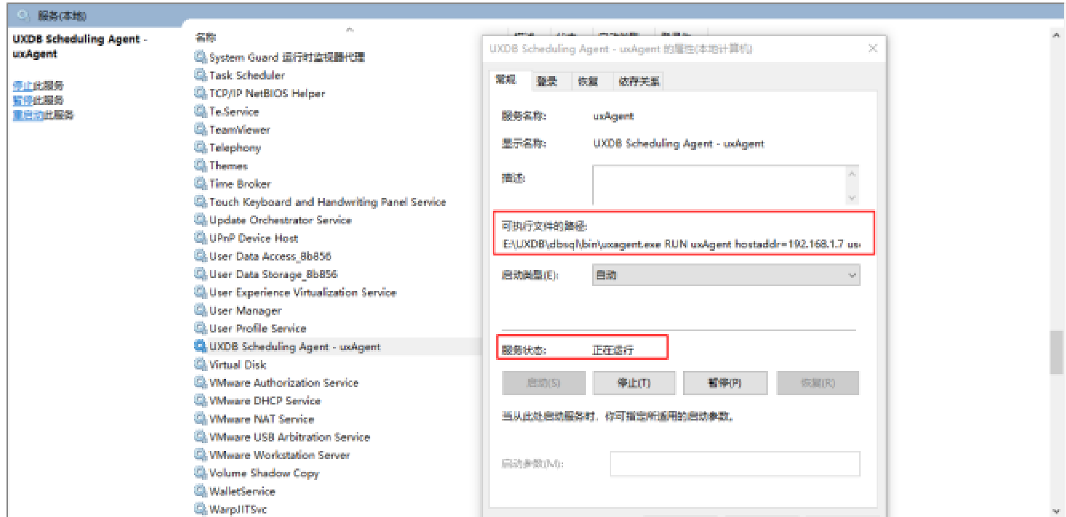
启动uxAgent报错。



解决方法：打开注册表编辑器WIN +R 输入：regedit；进入HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\uxAgent修改可执行文件的路径。



重新启动服务。



16.3. 示例

16.3.1. 创建定时任务

加载插件成功后，会在数据库中创建一个uxagent的schema，里面包含一些定时任务需要的接口和元数据表，创建一个新的job就是操作这些表，插入job的配置信息到对应的表中，如下所示。

```
uxdb=# \d
                关联列表
架构模式 |          名称           | 类型   | 拥有者
---------+-------------------------+--------+--------
uxagent | uxa_exception           | 数据表 | uxdb
uxagent | uxa_exception_jexid_seq | 序列数 | uxdb
uxagent | uxa_job                 | 数据表 | uxdb
uxagent | uxa_job_jobid_seq       | 序列数 | uxdb
uxagent | uxa_jobagent            | 数据表 | uxdb
uxagent | uxa_jobclass            | 数据表 | uxdb
uxagent | uxa_jobclass_jclid_seq  | 序列数 | uxdb
uxagent | uxa_joblog              | 数据表 | uxdb
uxagent | uxa_joblog_jlgid_seq    | 序列数 | uxdb
uxagent | uxa_jobstep             | 数据表 | uxdb
uxagent | uxa_jobstep_jstid_seq   | 序列数 | uxdb
uxagent | uxa_jobsteplog          | 数据表 | uxdb
uxagent | uxa_jobsteplog_jslid_seq | 序列数 | uxdb
uxagent | uxa_schedule            | 数据表 | uxdb
uxagent | uxa_schedule_jscid_seq  | 序列数 | uxdb
(15 行记录)
uxdb=# \df
                      函数列表
架构模式 |        名称          | 结果数据类型 | 参数数据类型 | 类型
---------+----------------------+--------------+--------------+---------
uxagent | uxa_exception_trigger | trigger      |              | 触发器
uxagent | uxa_is_leap_year     | boolean      | smallint     | 常规
uxagent | uxa_job_trigger      | trigger      |              | 触发器
uxagent | uxa_next_schedule    | integer, timestamp with time zone| integer, timestamp with
 time zone, timestamp with time zone, boolean[], boolean[], boolean[], boolean[],boolean[] | 常规
```

```
uxagent | uxa_schedule_trigger | trigger |  | 触发器
uxagent | uxagent_schema_version| smallint |  | 常规
(6 行记录)
```

在uxsql命令行中，可以操作uxAgent的元数据库表来添加一个定时任务。创建一个新的定时任务一般需要三个步骤。

1. 创建一个新的定时任务job；
2. 创建执行步骤steps；
3. 创建作业调度schedules。

下面是一个具体的示例，每分钟向test表中插入一条数据。

1. 创建job

   ```
   uxdb=# INSERT INTO uxagent.uxa_job(jobjclid, jobname, jobdesc, jobhostagent,
    jobenabled) VALUES (1::integer, 'job-test'::text, 'a example for job schedule within
    uxsql'::text, ''::text, true);
   INSERT 0 1
   ```

   jobjclid字段是本次创建的job的类型（job class），共有5种，请参见第 16.3.2 节 “元数据表”。

   jobhostagent字段指定运行job的主机，空表示所有主机都可运行，一般设置为空即可。

   查询jobid，在创建step和schedule时会使用到。

   ```
   uxdb=# SELECT jobid FROM uxa_job;
   jobid
   -------
    4
   (1 行记录)
   ```

2. 创建step

   ```
   uxdb=# INSERT INTO uxagent.uxa_jobstep(jstjobid,jstname,jstenabled,jstkind,jstconnstr,
   jstdbname,jstonerror,jstcode,jstdesc)
   VALUES (4,'step-for-job-test'::text,true,'s'::character(1), ''::text,'uxdb'::name,
    'f'::character(1),'insert into uxagent.test select now();'::text, 'the steps of job-test
    torun'::text);
   INSERT 0 1
   ```

   values从句中的值4，就是创建job步骤中查询的jobid。这个step执行如下命令。

   ```
   create table uxagent.test(a timestamp);
   insert into uxagent.test select now();
   ```

3. 创建作业调度

   ```
   uxdb=# INSERT INTO uxagent.uxa_schedule
   (jscjobid, jscname, jscdesc, jscenabled, jscstart, jscminutes, jschours, jscweekdays,
    jscmonthdays, jscmonths)
   VALUES (4, 'schedule-for-job-test'::text, 'a schedule of job-test'::text, true,'2019-08-29
    16:15:00+08'::timestamp with time zone,
   -- MinutesARRAY[true, true, true, true, true, true, true, true, true, true, true, true, true, true,
    true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true,
   ```

```
true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true,
true, true, true, true, true, true, true, true, true, true, true, true]::boolean[],
-- HoursARRAY[false, false, false, false, false, false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false, false, false, false]::boolean[],
-- Week daysARRAY[false, false, false, false, false, false, false]::boolean[],
-- Month daysARRAY[false, false, false, false, false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false,
false, false, false, false]::boolean[],
-- MonthsARRAY[false, false, false, false, false, false, false, false, false, false, false,
false]::boolean[]);
INSERT 0 1
```

同样，values从句的值4表示jobid。这里需要注意一下日期和时间字段，这是定时任务设置调度的部分，SQL中使用布尔类型的数组来表示日期和时间的各域，数组的长度就是各域的有效取值范围，如日期部分的month域，因为一年有12个月，所以其数组长度为12，其他域同理。true和false分别表示选择和不选在这个时间执行job。

执行结果。

```
uxdb=# select now();
 now
-------------------------------
2019-08-29 16:22:16.178407+08
(1 行记录)
uxdb=# select * from test;
 a
-------------------------------
2019-08-29 15:48:00.67718+08
2019-08-29 16:15:02.700994+08
2019-08-29 16:16:02.758783+08
2019-08-29 16:17:02.846406+08
2019-08-29 16:18:02.914731+08
2019-08-29 16:19:02.993701+08
2019-08-29 16:20:03.07522+08
2019-08-29 16:21:03.150455+08
2019-08-29 16:22:03.234137+08
(9 行记录)
```

16.3.2. 元数据表

- uxa_job

  记录job定义的元数据。

```
uxdb=# select * from uxa_job;
jobid | jobjclid | jobname | jobdesc | jobhostagent | jobenabled | jobcreated | jobchanged | jobagentid
 | jobnextrun| joblastrun
-------+----------+----------+----------------------------------------+--------------+------------
+---------------------------------------------
4 | 1 | job-test|a example for job schedule within uxsql |
 | t | 2019-08-29 15:54:06.89931+08 | 2019-08-29 15:54:06.89931+08 | |
 2019-08-29 16:36:00+08 | 2019-08-29 16:35:04.209332+08
(1 行记录)
```

- uxa_jobstep

记录每个job执行的step的元数据。

uxdb=# **select * from uxa_jobstep;**
jstid | jstjobid | jstname | jstdesc | jstenabled | jstkind | jstcode | jstconnstr | jstdbname
 | jstonerror| jscnextrun
-------+----------+----------+--+--------------+------------
+--
4 | 4 | step-for-job-test |the steps of job-test to run | t | s | insert into uxagent.test select now(); | |uxdb |
 f |
(1 行记录)

- uxa_schedule

记录每个job的调度的元数据。

uxdb=# **select * from uxa_schedule;**
jscid | jscjobid | jscname | jscdesc | jscenabled | jscstart | jscend | jscminutes | jschours | jscweekdays
 | jscmonthdays | jscmonths
-------+----------+----------+--+--------------+------------
+--
4 | 4 | schedule-for-job-test |a schedule of job-test | t | 2019-08-29 16:15:00+08 | |
 {t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,
t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t} |{f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f} | {f,f,f,f,f,f,f} | |
{f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f} | {f,f,f,f,f,f,f,f,f,f,f,f}
(1 行记录)

- uxa_jobagent

记录uxagent守护进程的信息。

uxdb=# **select * from uxa_jobagent;**
jagpid | jaglogintime | jagstation
--------+-------------------------------+-------------------
2143 | 2019-08-29 15:23:13.776622+08 | localhost
(1 行记录)

- uxa_jobclass

元数据，5种job类型。

uxdb=# **select * from uxa_jobagent;**
jclid | jclname
------+--------------------
 1 | Routine Maintenance
 2 | Data Import
 3 | Data Export
 4 | Data Summarisation
 5 | Miscellaneous
(5 行记录)

- uxa_joblog

每个job运行状态的日志。

uxdb=# **select * from uxa_joblog;**

```
jlgid | jlgjobid | jlgstatus | jlgstart |   jlgduration
------+----------+-----------+--------------+-----------------
  99 | 4 | s | 2019-08-29 16:15:02.672584+08 | 00:00:00.03865
 100 | 4 | s | 2019-08-29 16:16:02.737795+08 | 00:00:00.030618
 101 | 4 | s | 2019-08-29 16:17:02.820218+08 | 00:00:00.03422
 102 | 4 | s | 2019-08-29 16:18:02.893128+08 | 00:00:00.030145
 103 | 4 | s | 2019-08-29 16:19:02.973116+08 | 00:00:00.02985
 104 | 4 | s | 2019-08-29 16:20:03.053281+08 | 00:00:00.029872
 105 | 4 | s | 2019-08-29 16:21:03.129232+08 | 00:00:00.02937
 106 | 4 | s | 2019-08-29 16:22:03.206437+08 | 00:00:00.038498
 107 | 4 | s | 2019-08-29 16:23:03.279988+08 | 00:00:00.029533
 108 | 4 | s | 2019-08-29 16:24:03.360594+08 | 00:00:00.029433
 109 | 4 | s | 2019-08-29 16:25:03.427083+08 | 00:00:00.01847
 110 | 4 | s | 2019-08-29 16:26:03.541442+08 | 00:00:00.023226
 111 | 4 | s | 2019-08-29 16:27:03.60231+08  | 00:00:00.033018
 112 | 4 | s | 2019-08-29 16:28:03.687327+08 | 00:00:00.03721
 113 | 4 | s | 2019-08-29 16:29:03.760368+08 | 00:00:00.029912
 114 | 4 | s | 2019-08-29 16:30:03.841993+08 | 00:00:00.030117
 115 | 4 | s | 2019-08-29 16:31:03.923883+08 | 00:00:00.03032
......
......
......
```

- uxa_jobsteplog

  每个job的step的执行日志。

```
uxdb=# select * from uxa_jobsteplog;
jslid | jsljlgid | jsljstid | jslstatus | jslresult | jslstart | jslduration | jsloutput
-------+----------+-----------+------------------------------+----------------------+-------------------
99   | 99   | 4 | s | 1 | 2019-08-29 16:15:02.68332+08 | 00:00:00.024913 |
100  | 100  | 4 | s | 1 | 2019-08-2916:16:02.744372+08 | 00:00:00.021121 |
101  | 101  | 4 | s | 1 | 2019-08-29 16:17:02.82876+08 | 00:00:00.02325   |
102  | 102  | 4 | s | 1 | 2019-08-29 16:18:02.899348+08 | 00:00:00.020938 |
103  | 103  | 4 | s | 1 | 2019-08-2916:19:02.979277+08 | 00:00:00.019616 |
104  | 104  | 4 | s | 1 | 2019-08-2916:20:03.060626+08 | 00:00:00.019836 |
......
......
......
```

- uxa_exception

  记录job执行的异常信息。

```
uxdb=# select * from uxa_exception;
jexid | jexscid | jexdate | jextime
------+---------+---------+---------
(0 行记录)
```

第 17 章 uxaudit

UXDB提供了uxaudit插件,此插件主要用于创建审计员uxad、创建审计表uxaudit.session、uxaudit.logon、uxaudit.log_event、uxaudit.audit_statement、uxaudit.audit_substatement、uxaudit.audit_substatement_detail，并在插件初始化过程中创建DDL、DML、FUNCTION审计钩子函数，以此对后续用户连接数据库、操作数据库行为进行审计。

数据库服务端程序审计到的用户操作行为信息会自动保存至uxaudit.session、uxaudit.logon、uxaudit.log_event、uxaudit.audit_statement、uxaudit.audit_substatement、uxaudit.audit_substatement_detail审计表中，审计用户uxad可以通过查看vw_audit_event审计视图分析发现违规操作、异常行为、潜在侵害等。

17.1. 审计表

- log_event

  用于记录用户具体的操作事件信息。

表 17.1. log_event

段名	含义
session_id	登录的session的ID
session_line_num	在该session中执行的SQL的代号
log_time	日志时间
command	执行的SQL命令
error_severity	日志级别
sql_state_code	SQL执行完毕的状态码
virtual_transaction_id	虚拟事务ID
transaction_id	事务ID
message	描述
detail	详情
hint	该SQL执行的提示信息
query	查询
query_pos	查询的position
internal_query	内部查询
internal_query_pos	内部查询position
context	上下文
location	Location信息

- session

  用于记录用户登录会话信息。

表 17.2. session

段名	含义
process_id	进程ID
session_start_time	session开始时间
session_end_time	session结束时间
user_name	登陆用户名
role_name	登陆角色名
application_name	进程名字
connection_from	从何处进行的连接
client_mac	客户端mac地址
auth_method	客户端认证方式

- logon

用于记录用户登录信息。

表 17.3. logon

段名	含义
user_name	登录用户名
last_success	上一次成功登录时间
current_success	当前成功登陆的时间
last_failure	最后一次失败的时间
failures_since_last_success	从最后一次成功登陆时间之后，连续登陆失败的次数

- audit_statement

用于记录会话中SQL执行情况。

表 17.4. audit_statement

段名	含义
session_id	登陆的sessionID
statement_id	SQL statement ID
state	执行状态
error_session_line_num	错误的statement ID在当前session中是第几个SQL命令

- audit_substatement

用于记录会话中SQL详细信息（包含子SQL）。

表 17.5. audit_substatement

段名	含义
session_id	登陆的sessionID
statement_id	SQL statement ID
substatement_id	子SQL statement ID
substatement	SQL正文
parameter	SQL参数列表

- audit_substatement_detail

用于记录会话中SQL详细信息（包含子SQL），包含审计类型、分类、命令、客体类型及名称。

表 17.6. audit_substatement_detail

段名	含义
session_id	登陆的sessionID
statement_id	SQL statement ID
substatement_id	子SQL statement ID
session_line_num	在该session中执行的SQL的代号
audit_type	审计类型 session - 主体操作 或者object - 客体操作
class	分类
command	命令
object_type	客体类型
object_name	客体名称

17.2. 钩子函数

表 17.7. 钩子函数

函数	返回	描述
uxaudit_ProcessUtility_hook	Void无返回值	对DDL和utility commands命令执行会话审计，不审计substatements，所有子语句都由事件触发器覆盖审计。
uxaudit_ExecutorStart_hook	Void无返回值	对不包含表的查询文本和基本命令类型进行审计，包含SELECT、INSERT、UPDATE、DELETE操作。
uxaudit_ExecutorCheckPerms_hook	Bool类型	为DML执行会话（主体）和对象（客体）审计。
uxaudit_object_access_hook	Void无返回值	对非系统catalog下的函数调用操作进行审计。

17.3. 示例

1. 初始化带审计的集群。

   ./initdb -I -D test -W

2. 检查集群配置文件中是否包含审计插件。

   vi test/uxsino.conf

   查看其中配置项shared_preload_libraries中是否包含uxaudit。

3. 如果包含，则启动审计集群。

   ./ux_ctl -I -D test -l logfile start

> **注意**
>
> 该插件不需要手动执行create extension uxaudit。因为此插件已在initdb程序中被调用，不需要用户手动去创建。

第 18 章 uxcrypto

uxcrypto模块为UXDB提供了加解密函数。

使用uxcrypto模块前，首先需要执行CREATE EXTENSION命令。

```
CREATE EXTENSION uxcrypto;
```

18.1. 普通哈希函数

18.1.1. digest()

digest(data text, type text) returns bytea digest(data bytea, type text) returns bytea

计算给定data的二进制哈希值。type是要使用的算法。标准算法是md5、sha1、sha224、sha256、sha384和sha512。

如果想摘要成为一个十六进制字符串，可以在结果上使用encode()。如下所示。

```
CREATE OR REPLACE FUNCTION sha1(bytea) returns text AS $$
SELECT encode(digest($1, 'sha1'), 'hex')
$$ LANGUAGE SQL STRICT IMMUTABLE;
```

18.1.2. hmac()

hmac(data text, key text, type text) returns bytea hmac(data bytea, key text, type text) returns bytea

为带有密钥key的data计算哈希算法计算过的MAC。type与digest()中相同。

这与digest()相似，但是该哈希只能在知道密钥的情况下被重新计算出来。这阻止了非法用户修改数据且更改哈希以匹配。

如果该密钥大于哈希块的尺寸，它将先被哈希然后把结果用作密钥。

18.2. 口令哈希函数

函数crypt()和gen_salt()是特别设计用来做口令哈希的。crypt()完成哈希，而gen_salt()负责为前者准备算法参数。

crypt()中的算法在以下方面不同于通常的MD5或SHA1哈希算法。

1. 它们很慢。由于数据量很小，这是增加蛮力口令破解难度的唯一方法。
2. 它们使用一个随机值（称为salt），这样具有相同口令的用户将得到不同的密文口令。这也是针对逆转算法的一种额外保护。
3. 它们会在结果中包括算法类型，这样用不同算法哈希的口令能共存。
4. 有些算法是自适应的 — 这意味着当计算机变得更快时，可以调整该算法变得更慢，而不会产生与现有口令的不兼容。

表 18.1. crypt()支持的算法

算法	最大口令长度	是否自适应	Salt位数	输出长度	描述
bf	72	yes	128	60	基Blowfish，变体2a。
md5	无限制	no	48	34	基于MD5的加密。
xdes	8	yes	24	20	扩展的DES。
des	8	no	12	13	原生UNIX加密。

18.2.1. crypt()

crypt(password text, salt text) returns text

计算password的一个crypt(3) 风格的哈希。在存储一个新口令时，需要使用gen_salt()产生一个新的salt值。要检查一个口令，把存储的哈希值作为salt，并且测试结果是否匹配存储的值。

设置一个新口令的示例。

UPDATE ... SET pswhash = crypt('new password', gen_salt('md5'));

认证的示例。

SELECT (pswhash = crypt('entered password', pswhash)) AS pswmatch FROM ... ;

如果输入的口令正确，返回true。

18.2.2. gen_salt()

gen_salt(type text [, iter_count integer]) returns text

产生一个在crypt()中使用的随机salt字符串。该salt字符串告诉crypt()要使用哪种算法。

type参数指定哈希算法，可接受的类型是：des、xdes、md5以及bf。

iter_count参数可以为使用迭代计数的算法指定迭代计数。计数越高，哈希口令花的时间更长因而需要更多时间攻破。不过使用太高的计数会导致计算哈希的时间高达数年 — 这是不切实际的。如果iter_count参数被忽略，将使用默认的迭代计数。允许的iter_count值与算法相关，如下表所示。

表 18.2. crypt()的迭代计数

算法	默认值	最小值	最大值
xdes	725	1	16777215
bf	6	4	31

对xdes算法还有额外的限制：迭代计数必须是奇数。

要选取一个合适的迭代计数，最初DES加密被设计成在当时的硬件上每秒钟完成4次哈希，低于每秒4次哈希的速度很可能会损害可用性，而超过每秒100次哈希又可能太快了。

下表给出了不同哈希算法的相对慢度的综述。该表展示了在假设口令只含有小写字母或者大小写字母及数字的情况下，在一个8字符口令中尝试所有字符组合所需要的时间。在crypt-bf项中，斜线之后的数字是gen_salt的iter_count参数。

表 18.3. 哈希算法速度

算法	次哈希/秒	对于[a-z]	对于[A-Za-z0-9]	相对于md5 hash的持续时间
crypt-bf/8	1792	4 年	3927 年	100k
crypt-bf/7	3648	2 年	1929 年	50k
crypt-bf/6	7168	1 年	982 年	25k
crypt-bf/5	13504	188 天	521 年	12.5k
crypt-md5	171584	15 天	41 年	1k
crypt-des	23221568	157.5 分	108 天	7
sha1	37774272	90 分	68 天	4
md5 (hash)	150085504	22.5 分	17 天	1

注意

- 使用的机器是一台Intel Mobile Core i3。
- crypt-des和crypt-md5算法的数据是取自John the Ripper v1.6.38 -test。
- md5 hash的数据来自于mdcrack 1.2。
- sha1的数据来自于lcrack-20031130-beta。
- crypt-bf的数据是采用在1000个8字符口令上循环的简单程序采集到的。这种方法能展示不同迭代次数的速度。

注意“尝试所有组合”并非是实际中会采用的方式。通常口令破解都是在词典的帮助下完成的，词典中会包含常用词以及它们的多种变化。因此，甚至有些像词的口令被破解的时间可能会大大小于上面建议的数字。

18.3. PGP加密函数

函数实现了OpenPGP (RFC 4880) 标准的加密部分。对称密钥和公钥加密都支持。

一个加密的PGP消息由两个包组成。

- 包含会话密钥的包 — 加密过的对称密钥或者公钥。
- 包含用会话密钥加密过的数据的包。

使用对称密钥（即一个口令）加密。

1. 给定的口令被String2Key (S2K) 算法哈希。这更像crypt()算法 — 有目的地慢并且使用随机salt — 但是它会产生一个全长度的二进制密钥。
2. 如果要求一个独立的会话密钥，将会生成一个新的随机密钥。否则S2K密钥将被直接用作会话密钥。

3. 如果直接使用S2K密钥，那么只有S2K设置将被放入会话密钥包中，否则会话密钥会用S2K密钥加密并且放入会话密钥包中。

使用公钥加密。

1. 一个新的随机会话密钥会被生成。
2. 它被用公共密钥加密并且放入到会话密钥包中。

在两种情况下，要被加密的数据按下列步骤被处理。

1. 可选的数据操纵：压缩、转换成UTF-8或者行末转换。
2. 数据会被加上一个随机字节的块作为前缀。这等效于使用一个随机IV。
3. 追加一个随机前缀和数据的SHA1哈希。
4. 所有这些都用会话密钥加密并且放在数据包中。

18.3.1. pgp_sym_encrypt()

pgp_sym_encrypt(data text, psw text [, options text]) returns bytea pgp_sym_encrypt_bytea(data bytea, psw text [, options text]) returns bytea

使用对称PGP密钥psw加密data。

options参数包含下文所述的选项设置。

18.3.2. pgp_sym_decrypt()

pgp_sym_decrypt(msg bytea, psw text [, options text]) returns text pgp_sym_decrypt_bytea(msg bytea, psw text [, options text]) returns bytea

解密用对称密钥加密过的PGP消息。

不允许使用pgp_sym_decrypt解密bytea数据。这是为了避免输出非法的字符数据。可以使用pgp_sym_decrypt_bytea解密原始文本数据。

options参数可以包含下文所述的选项设置。

18.3.3. pgp_pub_encrypt()

pgp_pub_encrypt(data text, key bytea [, options text]) returns bytea pgp_pub_encrypt_bytea(data bytea, key bytea [, options text]) returns bytea

用一个公共PGP密钥key加密data。给这个函数一个私钥会产生错误。

options参数可以包含下文所述的选项设置。

18.3.4. pgp_pub_decrypt()

pgp_pub_decrypt(msg bytea, key bytea [, psw text [, options text]]) returns text pgp_pub_decrypt_bytea(msg bytea, key bytea [, psw text [, options text]]) returns bytea

解密一个公共密钥加密的消息。key必须是用来加密的公钥对应的私钥。如果私钥是用口令保护的，必须在psw中给出该口令。如果没有口令，但想指定选项，需要给出一个空口令。

不允许使用pgp_pub_decrypt解密bytea数据。这是为了避免输出非法的字符数据。可以使用pgp_pub_decrypt_bytea解密原始文本数据。

options参数可以包含下文所述的选项设置。

18.3.5. pgp_key_id()

pgp_key_id(bytea) returns text

pgp_key_id抽取一个PGP公钥或私钥的密钥ID。或者如果给定了一个加密过的消息，它将给出一个用来加密数据的密钥ID。

pgp_key_id能够返回2个特殊密钥ID。

- SYMKEY：该消息是用对称密钥加密的。
- ANYKEY：该消息是用公钥加密的，但是密钥ID已经被移除。这意味着需要尝试所有的密钥来看看哪个能解密该消息。uxcrypto本身不产生这样的消息。

> **注意**
>
> 不同的密钥可能具有相同的ID。这很少见但属于正常事件。客户端应用应尝试用每一个去解密，看看哪个合适 — 类似ANYKEY。

18.3.6. armor(), dearmor()

armor(data bytea [, keys text[], values text[]]) returns text dearmor(data text) returns bytea

这些函数把二进制数据封装/解包成PGP ASCII-armored格式，其基本上是带有CRC和额外格式化的Base64。

如果指定了keys和values数组，每一个键/值对的armored格式上会增加一个armor header。两个数组都必须是单一维度的，长度必须相同。键和值不能包含任何非ASCII字符。

18.3.7. pgp_armor_headers

pgp_armor_headers(data text, key out text, value out text) returns setof record

pgp_armor_headers()从data中抽取armor header。返回值是一个有两列的行集合，包括键和值。如果键或值包含任何非ASCII字符，会被视作UTF-8。

18.3.8. PGP函数的选项

选项被命名为与GnuPG类似的形式。一个选项的值应该在一个等号后给出，各个选项之间用逗号分隔。如下所示。

pgp_sym_encrypt(data, psw, 'compress-algo=1, cipher-algo=aes256')

除了convert-crlf之外所有这些选项只适用于加密函数。解密函数会从PGP数据中得到这些参数。

1. cipher-algo

   要用哪个密码算法。

值：bf，aes128，aes192，aes256（只用于OpenSSL：3des，cast5）

默认：aes128

适用于：pgp_sym_encrypt，pgp_pub_encrypt

2. convert-crlf

   加密时是否把\n转换成\r\n以及解密时是否把\r\n转换成\n。RFC 4880指定文本数据存储时应该使用\r\n换行。使用这个选项能够得到完全RFC兼容的行为。

   值：0，1

   默认：0

   适用于：pgp_sym_encrypt，pgp_pub_encrypt，pgp_sym_decrypt，pgp_pub_decrypt

3. disable-mdc

   不用SHA-1保护数据。使用这个选项的好处是实现与古董级别PGP产品的兼容，这些产品在受SHA-1保护的包被加入到RFC 4880之前就已经存在了。

   值：0，1

   默认：0

   适用于：pgp_sym_encrypt，pgp_pub_encrypt

4. sess-key

   使用单独的会话密钥。公钥加密总是使用一个单独的会话密钥。这个选项是用于对称密钥加密的，对称密钥加密默认直接使用S2K密钥。

   值：0，1

   默认：0

   适用于：pgp_sym_encrypt

5. s2k-mode

   值：

   0 - 不用salt。这种做法十分危险！

   1 - 用salt但是使用固定的迭代计数。

   3 - 可变的迭代计数。

   默认：3

   适用于：pgp_sym_encrypt

6. s2k-count

   S2K算法要使用的迭代次数。它必须是一个位于1024和65011712之间的值，首尾两个值包括在内。

默认：65536和253952之间的一个随机值。

适用于：pgp_sym_encrypt，只能用于s2k-mode=3。

7. s2k-digest-algo

   要在S2K计算中使用哪种摘要算法。

   值：md5，sha1

   默认：sha1

   适用于：pgp_sym_encrypt

8. s2k-cipher-algo

   要用哪种密码来加密独立的会话密钥。

   值：bf，aes，aes128，aes192，aes256

   默认：use cipher-algo

   适用于：pgp_sym_encrypt

9. unicode-mode

   是否把文本数据在数据库内部编码和UTF-8之间来回转换。如果数据库已经是UTF-8，将不会转换，但是消息将被标记为UTF-8。没有这个选项它将不会被标记。

   值：0，1

   默认：0

   适用于：pgp_sym_encrypt，pgp_pub_encrypt

18.3.9. 用GnuPG生成PGP密钥

- 生成一个新密钥

  gpg --gen-key

  更好的密钥类型是“DSA和Elgamal”。

  对于RSA密钥，必须创建仅用于签名的DSA或RSA密钥作为主控密钥，然后用gpg --edit-key增加一个RSA加密子密钥。

- 列举密钥

  gpg --list-secret-keys

- 以ASCII-armor格式导出一个公钥

  gpg -a --export KEYID > public.key

- 以ASCII-armor格式导出一个私钥

```
gpg -a --export-secret-keys KEYID > secret.key
```

在把这些密钥交给PGP函数之前，需要对它们使用dearmor()。或者如果能处理二进制数据，可以从命令中去掉-a。

18.3.10. PGP代码的限制

- 不支持签名。这也意味着它不检查加密子密钥是否属于主控密钥。
- 不支持加密密钥作为主控密钥。由于通常并不鼓励那种用法，这应该不属于严重的问题。
- 不支持多个子密钥。在实践中普遍需要多个子密钥。并且，不能把常规GPG/PGP密钥用于uxcrypto，而是创建一些新的密钥，因为使用场景不同。

18.4. 原始的加密函数

这些函数只在数据上运行一次加密，不具有PGP加密的任何先进特性。因此它们有一些主要的问题。

1. 直接把用户密钥用作加密密钥。
2. 不提供任何完整性检查来查看被加密数据是否被修改。
3. 希望用户自己管理所有加密参数，甚至是IV。
4. 无法处理文本。

因此，在介绍了PGP加密后，不鼓励使用原始的加密函数。

encrypt(data bytea, key bytea, type text) returns bytea decrypt(data bytea, key bytea, type text) returns bytea encrypt_iv(data bytea, key bytea, iv bytea, type text) returns bytea decrypt_iv(data bytea, key bytea, iv bytea, type text) returns bytea

使用type指定的密码方法加密/解密数据。type字符串的语法如下所示。

algorithm [- mode] [/pad: padding]

表 18.4. type字符串参数说明

名称	可选值	描述
algorithm	bf	Blowfish
	aes	AES (Rijndael-128)
mode	cbc	下一个块依赖前一个（默认）
	ecb	每一个块被独立加密（只用于测试）
padding	pkcs	数据可以是任意长度（默认）
	none	数据必须是密码块尺寸的倍数

因此，下面的示例是等效的。

encrypt(data, 'fooz', 'bf')
encrypt(data, 'fooz', 'bf-cbc/pad:pkcs')

在encrypt_iv和decrypt_iv中，iv参数是CBC模式的初始值，ECB会忽略它。如果不是准确的块尺寸，它会被修剪或填充为零。在没有这个参数的函数中，它的值都被默认为零。

18.5. 随机数据函数

gen_random_bytes(count integer) returns bytea

返回count个强随机字节。一次最多可以抽取1024个字节，避免耗尽随机数生成池。

gen_random_uuid() returns uuid

返回一个版本4的随机UUID。

18.6. 注解

18.6.1. NULL处理

按照SQL中的标准，只要任何参数是NULL，所有的函数都会返回NULL。在使用不当时这可能会导致安全风险。

18.6.2. 安全性限制

所有uxcrypto函数都在数据库服务器内部运行。这意味着在uxcrypto和客户端应用之间传输的所有数据和口令都是明文。

1. 本地连接或者使用SSL连接。

2. 信任系统管理员和数据库管理员。

如果不能这样做，那么最好在客户端应用中进行加密。

该实现无法抵抗侧信道攻击。例如，一个uxcrypto解密函数完成所需的时间是随着密文尺寸变化的。

第 19 章 uxdb_fdw

uxdb_fdw模块提供外部数据封装器uxdb_fdw，用来访问存储在外部UXDB服务器中的数据。

这个模块提供的功能大体上覆盖了dblink模块的功能。但是uxdb_fdw提供了更透明且更符合标准的语法来访问远程表，并且可以在很多情况下提供更好的性能。

使用uxdb_fdw来为远程访问做准备，如下所示。

1. 使用CREATE EXTENSION来安装uxdb_fdw扩展。

2. 使用CREATE SERVER创建一个外部服务器对象，表示想连接的远程数据库。指定除了user和password之外的连接信息作为该服务器对象的选项。

3. 使用CREATE USER MAPPING创建一个用户映射，每一个用户映射都代表允许一个数据库用户访问一个外部服务器。指定远程用户名和口令作为用户映射的user和password选项。

4. 为每一个要访问的远程表使用CREATE FOREIGN TABLE或者IMPORT FOREIGN SCHEMA创建一个外部表。外部表的列必须匹配被引用的远程表。但是，如果在外部表对象的选项中指定了正确的远程对象名称，可以使用不同于远程表的表名和或列名。

做好远程访问的准备后，只需要从一个外部表SELECT来访问存储在它的底层的远程表中的数据。也可以使用INSERT、UPDATE或DELETE修改远程表（当然，用户映射中已经指定的远程用户必须具有这些权限）。

> **注意**
>
> 当前uxdb_fdw缺少对于带ON CONFLICT DO UPDATE子句的INSERT语句的支持。不过，如果省略了唯一的索引推断规范，它支持ON CONFLICT DO NOTHING子句。

通常推荐一个外部表的列被声明为与被引用的远程表列完全相同的数据类型和排序规则。尽管uxdb_fdw目前能够在需要时执行数据类型转换，但是当类型或排序规则不匹配时可能会发生语义异常，因为远程服务器解释WHERE子句时可能会与本地服务器有所不同。

> **注意**
>
> 一个外部表可以被声明比底层的远程表较少的列，或者使用不同的列序。与远程表的列匹配是通过名称而不是位置进行的。

19.1. fdw选项

19.1.1. 连接选项

一个使用uxdb_fdw外部数据封装器的外部服务器可以使用libpq在连接字符串中能接受的选项，不允许以下选项。

- user和password（在用户映射中指定）。

- client_encoding（自动从本地服务器编码设置）。

- fallback_application_name（总是设置为uxdb_fdw） 。

只有系统管理员可以在不经过口令认证的情况下连接到外部服务器，因此应为属于非系统管理员的用户映射指定password选项。

19.1.2. 对象名称选项

对象名称选项控制被发送到远程UXDB服务器的SQL语句中使用的名称。当一个外部表被使用不同于底层远程表的名称创建时，就需要这些选项。

schema_name

　这个选项给出用在远程服务器之上的外部表的模式名称，可以为一个外部表指定该选项。如果这个选项被忽略，该外部表的模式名称将被使用。

table_name

　这个选项给出用在远程服务器上的外部表表名，可以为一个外部表指定该选项。如果这个选项被忽略，该外部表的名称将被使用。

column_name

　这个选项给出用在远程服务器上列的列名，可以为一个外部表指定该选项。如果这个选项被忽略，该列的名称将被使用。

19.1.3. 代价估计选项

uxdb_fdw通过在远程服务器上执行查询来检索远程数据，因此理想的扫描一个外部表的估计代价应该是在远程服务器上完成它的花销，外加一些通信开销。得到这样一个估计的最可靠的方法是询问远程服务器并加上一些通信开销 — 但是对于简单查询，不值得为获得一个代价估计而额外使用一次远程查询。因此uxdb_fdw提供了下列选项来控制如何完成代价估计。

use_remote_estimate

　这个选项控制uxdb_fdw是否发出EXPLAIN命令来获得代价估计，可以为一个外部表或一个外部服务器指定该选项。外部表的设置会覆盖它的服务器的任何设置，但是只用于这个表。默认值是false。

fdw_startup_cost

　这个选项是要被添加到服务器上任何外部表扫描的估计启动代价的数值，可以为一个外部服务器指定该选项。这表示建立远端连接、解析和规划查询等额外负荷。默认值是100。

fdw_tuple_cost

　这个选项是被用作服务器上外部表扫描的每个元组额外代价的一个数值，可以为一个外部服务器指定该选项。这表示在服务器之间数据传输的额外负荷。可以增加或减少这个数来反映到远程服务器更高或更低的网络延迟。默认值是0.01。

当use_remote_estimate为真时，uxdb_fdw从远程服务器获得行计数和代价估计，然后在代价估计上加上fdw_startup_cost和fdw_tuple_cost。

当use_remote_estimate为假时，uxdb_fdw执行本地行计数和代价估计，并且接着在代价估计上加上fdw_startup_cost 和fdw_tuple_cost。这种本地估计不会很准确，除非有远程表统计数据的本地拷贝可用。在外部表上运行ANALYZE是更新本地统计数据的方法，这将执行远程表的一次

扫描并接着计算和存储统计数据，就好像表在本地一样。保留本地统计数据可能是一种有用的方法来减少一个远程表的预查询规划负荷 — 但是如果远程表被频繁更新，本地统计数据将很快就被废弃。

19.1.4. 远程执行选项

默认情况下，只有使用了内建操作符和函数的WHERE子句才会被考虑在远程服务器上执行。涉及非内建函数的WHERE子句将会在取完行后在本地进行检查。如果这类函数在远程服务器上可用并且可以用来产生和本地执行时一样的结果，则可以通过将这种WHERE子句发送到远程执行来提高性能。可以用如下的选项控制这种行为。

extensions

这个选项是一个用逗号分隔的已安装的UXDB扩展名称列表，这些扩展在本地和远程服务器上具有兼容的版本。属于该列表中扩展的不可变函数和操作符将被考虑转移到远程服务器上执行。这个选项只能由外部服务器指定，无法逐个表指定。

在使用extensions选项时，用户应该负责确保列出的扩展在本地和远程服务器上都存在且保持一致。否则，远程查询可能失败或者行为异常。

fetch_size

这个选项指定在每次获取行的操作中uxdb_fdw应该得到的行数。可以为一个外部表或一个外部服务器指定该选项。在表上指定的选项将会覆盖在服务器级别上指定的选项。默认值为100。

19.1.5. 可更新性选项

默认情况下，所有使用uxdb_fdw的外部表都被假定是可更新的。可以使用以下选项覆盖此选项。

updatable

这个选项控制uxdb_fdw是否允许外部表被使用INSERT、UPDATE和DELETE命令更新。可以为一个外部表或一个外部服务器指定该选项。一个表级选项会覆盖一个服务器级选项。默认值是true。

当然，如果远程表实际上并非可更新的，将产生一个错误。这个选项的使用主要是允许在不查询远程服务器的情况下在本地抛出错误。但是要注意information_schema视图根据这个选项的设置报告uxdb_fdw外部表是可更新的（或者不可更新的），而不经过远程服务器的任何检查。

19.1.6. 导入选项

uxdb_fdw能使用IMPORT FOREIGN SCHEMA导入外部表定义。这个命令会在本地服务器上创建外部表定义，这个定义能匹配存在于远程服务器上的表或者视图。如果要被导入的远程表有用户自定义数据类型的列，本地服务器上也必须具有相同名称的兼容类型。

导入行为可以用下列选项自定义（在IMPORT FOREIGN SCHEMA命令中给出）。

import_collate

这个选项控制是否在从外部服务器导入的外部表定义中包括列的COLLATE选项。默认是true。如果远程服务器具有和本地服务器不同的排序规则名集合，可能需要关闭这个选项，在不同的操作系统上运行时很可能就是这样。

import_default

这个选项控制是否在从外部服务器导入的外部表定义中包括列的DEFAULT表达式。默认是false。如果启用这个选项，要当心在远程服务器和本地服务器上计算表达式的方式不同，nextval()常会导致这类问题。如果导入的默认值表达式使用了一个本地不存在的函数或者操作符，IMPORT将整个失败。

import_not_null

这个选项控制是否在从外部服务器导入的外部表定义中包括列的NOT NULL约束。默认是true。

> **注意**
>
> 除NOT NULL之外的约束将不会从远程表中导入。虽然UXDB确实支持外部表上的CHECK约束，但不会自动导入它们，因为约束表达式在本地和远程服务器上的计算可能会不同。CHECK约束中的任何这类不一致都可能导致查询优化中很难检测的错误。因此，如果希望导入CHECK约束，必须手工完成，并且应仔细验证每一个这种约束的语义。
>
> 自动排除作为其他表的分区的表或外部表。分区表的父表被导入，除非它们是其他表的分区。 由于所有数据都可以通过作为分区层次根的分区表来访问，所以这种方法应该允许访问所有数据而不创建额外的对象。

19.2. 连接管理

uxdb_fdw在第一个使用关联到外部服务器的外部表的查询期间建立一个到外部服务器的连接。这个连接会被保持，并被重用于同一个会话中的后续查询。但是，如果使用了多个用户实体（用户映射）来访问外部服务器，会为每一个用户映射建立一个连接。

19.3. 事务管理

在一个引用外部服务器上任何远程表的查询期间，如果还没有根据当前的本地事务打开一个远程事务，uxdb_fdw将在远程服务器上打开该事务。当本地事务提交或中止时，远程事务也被提交或中止。保存点也采用创建相应的远程保存点来管理。

当本地事务为SERIALIZABLE隔离级别时，远程事务使用SERIALIZABLE隔离级别；否则远程事务使用REPEATABLE READ隔离级别。这种选择保证当一个查询在远程服务器上执行多个表查询，能够为所有扫描得到快照一致的结果。即便由于其他活动在远程服务器上发生了其他并发更新，在单一事务中的后继查询将会看到来自远程服务器的相同数据。如果本地事务使用SERIALIZABLE或REPEATABLE READ隔离级别，这种行为也是可以预期的。

19.4. 远程查询优化

uxdb_fdw尝试优化远程查询来减少从外部服务器传来的数据量。这可以通过把查询的WHERE子句发送给远程服务器执行来完成，并且还可以不检索当前查询不需要的表列。为了降低查询被误执行的风险，除非WHERE子句使用的数据类型、操作符和函数都是内建的或者属于列在该外部服务器的extensions选项中的一个扩展，否则将不会把WHERE子句发送到远程服务器。这些子句中的操作符合函数也必须是IMMUTABLE。对于UPDATE或者DELETE查询，如果没有不能发送给远程服务器的WHERE子句、没有查询的本地连接、目标表上没有本地的行级BEFORE或AFTER触发器，没有来自父视图的CHECK OPTION约束，uxdb_fdw会尝试通过将整个查询发送给远程服务器来优化查询的

执行。在UPDATE中，赋值给目标列的表达式只能使用内建数据类型、IMMUTABLE操作符，这样能降低查询被误执行的风险。

当uxdb_fdw碰到同一个外部服务器上的外部表之间的连接时，它会把整个连接发送给外部服务器，除非由于某些原因它认为逐个从每一个表取得行的效率更高或者涉及的表引用属于不同的用户映射。在发送JOIN子句时，它也会采取和上述WHERE子句相同的预防措施。

实际被发送到远程服务器执行的查询可以使用EXPLAIN VERBOSE来检查。

19.5. 远程查询执行环境

在uxdb_fdw开启的远程会话中，search_path参数只被设置为ux_catalog，因此只有内建对象可以在无模式限定时可见。这对于uxdb_fdw本身产生的查询来说不是问题，因为它总是会提供这样的限定。不过，这可能会对在远程服务器上通过触发器或者远程表上的规则执行的函数带来灾难。例如，如果一个远程表实际是一个视图，任何在该视图中使用的函数都将被在这个受限的搜索路径中执行。推荐在这类函数中用模式限定所有名称，或者为这类函数附着SET search_path选项来建立它们所期望的搜索路径环境。

uxdb_fdw同样为各种参数建立远程会话设置，如下所示。

- TimeZone设置为UTC。
- DateStyle设置为ISO。
- IntervalStyle设置为uxdb。
- extra_float_digits设置为3。

如果需要也可以使用函数SET选项来处理。

不推荐通过更改这些参数的会话级设置来覆盖此行为，这很可能会导致uxdb_fdw故障。

19.6. 示例

这里是一个用uxdb_fdw创建外部表的示例。

1. 首先安装该扩展。

```
CREATE EXTENSION uxdb_fdw;
```

2. 使用CREATE SERVER创建一个外部服务器（示例中连接的UXDB服务器主机IP：192.83.123.89；端口：5432；数据库：foreign_db）。

```
CREATE SERVER foreign_server
FOREIGN DATA WRAPPER uxdb_fdw
OPTIONS (host '192.83.123.89', port '5432', dbname 'foreign_db');
```

3. 用CREATE USER MAPPING定义一个用户映射来标识在远程服务器上使用哪个角色。

```
CREATE USER MAPPING FOR local_user
 SERVER foreign_server
OPTIONS (user 'foreign_user', password 'password');
```

4. 使用CREATE FOREIGN TABLE创建外部表（示例中访问远程服务器上的表名。

//some_schema.some_table；它的本地名称：foreign_table）：

```
CREATE FOREIGN TABLE foreign_table (
id integer NOT NULL,
data text
)
SERVER foreign_server
OPTIONS (schema_name 'some_schema', table_name 'some_table');
```

CREATE FOREIGN TABLE中声明的列数据类型和其他特性必须要匹配实际的远程表。列名也必须匹配，不过也可以为个别列附上column_name选项以表示它们在远程服务器上对应哪个列。在很多情况中，要使用IMPORT FOREIGN SCHEMA手工构造外部表定义。

第 20 章 uxgis

使用uxgis之前要安装依赖库文件，依赖库包包括：json-c、gdal、geos、xml2、proj、CGAL（被SFCGAL依赖）、SFCGAL。

20.1. 依赖库

依赖库文件版本与系统环境关系密切，而且依赖文件数量繁多，如果全部通过rpm命令安装的话耗时耗力且不容易成功，故建议使用yum安装。安装步骤如下。

1. 安装epel源

   ```
   rpm -Uvh https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
   ```

2. 安装pg源

   ```
   yum install https://download.postgresql.org/pub/repos/yum/reporpms/EL-7-x86_64/pgdg-redhat-repo-latest.noarch.rpm
   ```

3. 安装scl源

   ```
   yum install centos-release-scl
   ```

4. yum安装依赖

   ```
   yum install json-c
   yum install gdal
   yum install xml2
   yum install proj
   yum install boost-serialization
   yum install mesa-libGLU
   yum install CGAL
   yum install SFCGAL
   yum install geos38
   ```

5. 验证UXGIS扩展是否安装完全

   创建并启动数据库，创建如下扩展，确认UXGIS扩展环境已完全搭建。

   ```
   CREATE EXTENSION UXGIS;
   CREATE EXTENSION fuzzystrmatch;
   CREATE EXTENSION UXGIS_sfcgal;
   CREATE EXTENSION UXGIS_topology;
   CREATE EXTENSION UXGIS_tiger_geocoder;
   CREATE EXTENSION address_standardizer;
   CREATE EXTENSION address_standardizer_data_us;
   ```

20.2. 示例

1. 准备空间数据库相关的.shp文件放到demo文件夹中备用。

.shp 文件中存的是图形的信息。

.dbf 文件中存的是属性的信息。

.prj 文件中存的是坐标的信息。

.sbn和.sbx 文件中存的空间索引。

```
[uxdb@localhost demo]$ pwd
/home/uxdb/Desktop/demo
[uxdb@localhost demo]$ ls
hyd1_4l.dbf  hyd1_4l.shp  hyd1_4l.shx  hyd1_4p.dbf  hyd1_4p.shp  hyd1_4p.shx
[uxdb@localhost demo]$
```

在此以导入.shp文件为例。

> **注意**
>
> 要使用uxgis的.shp文件转换功能，必须要安装uxgis安装包（可找优炫技术人员获取），否则使用过程中会报找不到libwgeom-2.4.so.0库错误，如下图示。
>
> ```
> [uxdb@localhost bin]$ pgsql2shp --help
> pgsql2shp: error while loading shared libraries: liblwgeom-2.4.so.0: cannot open
>  shared object file: No such file or directory
> ```
>
> 建议将安装包安装到默认路径下，因为uxgis编译安装是在默认路径下进行的。

2. Shape数据转换为sql文件。

   进入uxdbinstall/dbsql//bin目录下，创建demo文件夹，使用如下命令进行转换。

   shp2pgsql -s 3857 -c -W "GBK" /home/uxdb/Desktop/demo/hyd1_4l.shp >demo/hyd1_4l.sql

```
[uxdb@localhost bin]$ mkdir demo
[uxdb@localhost bin]$ shp2pgsql -s 3857 -c -W "GBK" /home/uxdb/Desktop/demo/hyd1
_4l.shp >demo/hyd1_4l.sql
Field fnode_ is an FTDouble with width 11 and precision 0
Field tnode_ is an FTDouble with width 11 and precision 0
Field lpoly_ is an FTDouble with width 11 and precision 0
Field rpoly_ is an FTDouble with width 11 and precision 0
Field length is an FTDouble with width 12 and precision 3
Field hyd1_4m_ is an FTDouble with width 11 and precision 0
Field hyd1_4m_id is an FTDouble with width 11 and precision 0
Shapefile type: Arc
Postgis type: MULTILINESTRING[2]
[uxdb@localhost bin]$
```

参数说明:

- -s 代表指定数据的SRID
- -c 代表数据将新建一个表
- -d 删除旧的表，重新建表并插入数据
- -a 向现有表中追加数据

- -p 仅创建表结构，不添加数据
- -W Shape文件中属性的字符集

  有时Shape数据中的字符集是其他，就可能报“Unable to convert data value to UTF-8 (iconv reports "无效或不完整的多字节字符或宽字符"). Current encoding is "UTF-8". Try "LATIN1" (Western European)”错误，这时候指定正确的字符集即可解决问题。

3. 建立空间数据库，并创建UXGIS相关扩展。

   a. 进入uxdbinstall/dbsql/bin目录下，创建数据库。

      initdb -W -D testdb

   b. 修改配置文件testdb/uxsinodb.conf，使其能进行uxgis扩展。

      去掉shared_preload_libraries的注释，进行修改。

      shared_preload_libraries = 'postgres_adaptor'   # (change requires restart)

      设置adaptor_pg_ux_mode_switch。

      adaptor_pg_ux_mode_switch = on

   c. 使用uxdb命令启动数据库，并通过uxsql连接。

      uxdb -D testdb

```
creating subdirectories ... ok
selecting default max_connections ... 100
selecting default shared_buffers ... 128MB
selecting dynamic shared memory implementation ... posix
creating configuration files ... ok
running bootstrap script ... ok
performing post-bootstrap initialization ... ok
syncing data to disk ... ok

Success. You can now start the database server using:

    uxdb -D testdb
or
    ux_ctl -D testdb -l logfile start
and
start client using:

    uxsql -d uxdb
[uxdb@localhost bin]$
```

   d. 创建uxgis扩展。

      CREATE ROLE gisdb;
      CREATE DATABASE shp2pgsqldemo WITH OWNER=gisdb;
      \c shp2pgsqldemo;
      CREATE EXTENSION uxgis;

CREATE EXTENSION fuzzystrmatch;
CREATE EXTENSION uxgis_sfcgal;
CREATE EXTENSION uxgis_topology;
CREATE EXTENSION uxgis_tiger_geocoder;
CREATE EXTENSION address_standardizer;
CREATE EXTENSION address_standardizer_data_us;

```
uxdb=# CREATE ROLE  gisdb;
CREATE ROLE
uxdb=# CREATE DATABASE shp2pgsqldemo WITH OWNER=gisdb;
CREATE DATABASE
uxdb=# \c shp2pgsqldemo;
You are now connected to database "shp2pgsqldemo" as user "uxdb".
shp2pgsqldemo=# CREATE EXTENSION postgis;
CREATE EXTENSION
shp2pgsqldemo=# CREATE EXTENSION fuzzystrmatch;
CREATE EXTENSION
shp2pgsqldemo=# CREATE EXTENSION postgis_sfcgal;
CREATE EXTENSION
shp2pgsqldemo=# CREATE EXTENSION postgis_topology;
CREATE EXTENSION
shp2pgsqldemo=# CREATE EXTENSION postgis_tiger_geocoder;
CREATE EXTENSION
shp2pgsqldemo=# CREATE EXTENSION address_standardizer;
CREATE EXTENSION
shp2pgsqldemo=# CREATE EXTENSION address_standardizer_data_us;
CREATE EXTENSION
shp2pgsqldemo=#
```

4. 使用uxsql向数据库导入使用Shape数据生成的.sql文件。

uxsql -d shp2pgsqldemo -f demo/hyd1_4l.sql -W

```
[uxdb@localhost bin]$ pwd
/home/uxdb/uxdbinstall/dbsql/bin
[uxdb@localhost bin]$ uxsql -d shp2pgsqldemo -f demo/hyd1_4l.sql -W
Password:
SET
SET
BEGIN
CREATE TABLE
ALTER TABLE
                    addgeometrycolumn
------------------------------------------------------------
 public.hyd1_4l.geom SRID:3857 TYPE:MULTILINESTRING DIMS:2
(1 row)

INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
```

导入成功，结果展示：

```
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
COMMIT
ANALYZE
[uxdb@localhost bin]$
```

5. 验证数据。

```
\d
SELECT COUNT(*) FROM hyd1_4l;
SELECT COUNT(*) FROM spatial_ref_sys;
SELECT * FROM us_lex;
```

```
shp2pgsqldemo=# SELECT COUNT(*) FROM hyd1_4l;
 count
-------
   840
(1 row)

shp2pgsqldemo=# SELECT COUNT(*) FROM spatial_ref_sys;
 count
-------
  5757
(1 row)

shp2pgsqldemo=# SELECT * FROM us_lex;
  id  | seq |          word          |         stdword          | token | is_c
ustom
------+-----+------------------------+--------------------------+-------+-----
------
    1 |   1 | #                      | #                        |    16 | f
    2 |   2 | #                      | #                        |     7 | f
    3 |   1 | &                      | AND                      |    13 | f
    4 |   2 | &                      | AND                      |     1 | f
    5 |   3 | &                      | AND                      |     7 | f
    6 |   1 | -                      | -                        |     9 | f
```

第 21 章 uxrowlocks

uxrowlocks模块提供了一个函数来显示一个指定表的行锁定信息。

默认情况下，使用仅限于系统管理员、ux_stat_scan_tables角色的成员和在该表上拥有SELECT权限的用户。

使用uxrowlocks模块前，首先需要执行CREATE EXTENSION命令，如下所示。

CREATE EXTENSION uxrowlocks;

21.1. 概述

uxrowlocks(text) returns setof record

参数是一个表的名称。结果是一个记录集合，其中每一行对应表中一个被锁定的行。输出列如下表所示。

表 21.1. uxrowlocks输出列

名称	类型	描述
locked_row	tid	被锁定行的元组ID（TID）。
locker	xid	持锁者的事务ID，如果是多事务则为多事务ID。
multi	boolean	如果持锁者是一个多事务，则为真。
xids	xid[]	持锁者的事务ID（如果是多事务则多于一个）。
lock_type	text[]	持锁者的锁模式（如果是多事务则多于一个），是一个Key Share、Share、For No Key Update、No Key Update、For Update、Update组成的数组。
pids	integer[]	锁定后端的进程ID（如果是多事务则多于一个）。

uxrowlocks会为目标表加AccessShareLock并且一个一个读取每一行来收集行的锁定信息。这对于一个大表速度较慢。

注意

1. 如果表被其他人整体加上了排他锁，uxrowlocks将被阻塞。
2. uxrowlocks不保证能产生一个自我一致的快照。在它执行期间，有可能加上一个新行锁，也有可能有旧行锁被释放。

uxrowlocks不显示被锁定行的内容。如果想同时查看行内容，可以进行如下操作。

```
SELECT * FROM accounts AS a, uxrowlocks('accounts') AS p
 WHERE p.locked_row = a.ctid;
```

> **注意**
>
> 不过要注意，这样查询将非常低效。

21.2. 示例

```
test=# SELECT * FROM uxrowlocks('t1');
 locked_row | lock_type | locker | multi |   xids    |     pids
------------+-----------+--------+-------+-----------+---------------
      (0,1) | Shared    |     19 | t     | {804,805} | {29066,29068}
      (0,2) | Shared    |     19 | t     | {804,805} | {29066,29068}
      (0,3) | Exclusive |    804 | f     | {804}     | {29066}
      (0,4) | Exclusive |    804 | f     | {804}     | {29066}
(4 rows)
```

第 22 章 uxstattuple

uxstattuple模块提供多种函数来获得元组层的统计信息。

由于这些函数返回详细的页面级信息，因此只有系统管理员在安装时才具有EXECUTE特权。在安装函数之后，用户可以发出GRANT命令来更改函数的权限，以允许非系统管理员用户执行它们。默认情况下，ux_stat_scan_tables角色的成员被授予访问权限。

使用uxstattuple模块前，首先需要执行CREATE EXTENSION命令，如下所示。

CREATE EXTENSION uxstattuple;

22.1. uxstattuple(regclass) returns record

uxstattuple返回一个关系的物理长度、“死亡”元组的百分比以及其他信息。这可以帮助用户决定是否需要清理。参数是目标关系的名称（可以有选择地用模式限定）或者OID。如下所示。

```
test=> SELECT * FROM uxstattuple('ux_catalog.ux_proc');
-[ RECORD 1 ]------+-------
table_len          | 458752
tuple_count        | 1470
tuple_len          | 438896
tuple_percent      | 95.67
dead_tuple_count   | 11
dead_tuple_len     | 3157
dead_tuple_percent | 0.69
free_space         | 8932
free_percent       | 1.95
```

表 22.1. uxstattuple输出列

列	类型	描述
table_len	bigint	物理关系长度，以字节计。
tuple_count	bigint	存活元组的数量。
tuple_len	bigint	存活元组的总长度，以字节计。
tuple_percent	float8	存活元组的百分比。
dead_tuple_count	bigint	死亡元组的数量。
dead_tuple_len	bigint	死亡元组的总长度，以字节计。
dead_tuple_percent	float8	死亡元组的百分比。
free_space	bigint	空闲空间总量，以字节计。
free_percent	float8	空闲空间的百分比。

注意

table_len将总是比tuple_len、 dead_tuple_len和free_space的和要大。这种差异是由固定的页面开销、每页指向元组的指针表以及确保元组正确对齐的填充造成的。

> uxstattuple只要求在关系上的一个读锁。因此结果不能反映一个即时快照，并发更新将影响结果。
>
> 如果HeapTupleSatisfiesDirty返回假，uxstattuple就判定一个元组是“死亡的”。

22.2. uxstatindex(regclass) returns record

uxstatindex返回B-tree索引的信息的记录。如下所示。

```
test=> SELECT * FROM uxstatindex('ux_cast_oid_index');
-[ RECORD 1 ]------+------
version            | 2
tree_level         | 0
index_size         | 16384
root_block_no      | 1
internal_pages     | 0
leaf_pages         | 1
empty_pages        | 0
deleted_pages      | 0
avg_leaf_density   | 54.27
leaf_fragmentation | 0
```

表 22.2. uxstatindex输出列

列	类型	描述
version	integer	B-tree版本号。
tree_level	integer	根页的树层次。
index_size	bigint	以字节计的索引总尺寸。
root_block_no	bigint	根页的位置（如果没有则为零）。
internal_pages	bigint	“内部”（上层）页面的数量。
leaf_pages	bigint	叶子页的数量。
empty_pages	bigint	空页的数量。
deleted_pages	bigint	删除页的数量。
avg_leaf_density	float8	叶子页的平均密度。
leaf_fragmentation	float8	叶子页碎片。

报告的index_size通常对应于internal_pages + leaf_pages + empty_pages + deleted_pages加一，因为它还包括索引的元页。

uxstattuple的结果是一页一页累计的并不表示整个索引的一个即时快照。

22.3. uxstatginindex(regclass) returns record

uxstatginindex返回GIN索引的信息的记录。如下所示。

创建GIN索引，请参见btree_gin示例

```
test=> SELECT * FROM uxstatginindex('test_gin_index');
-[ RECORD 1 ]--+--
version        | 1
pending_pages  | 0
pending_tuples | 0
```

表 22.3. uxstatginindex输出列

列	类型	描述
version	integer	GIN版本号。
pending_pages	integer	待处理列表中的页面数。
pending_tuples	bigint	待处理列表中的元组数。

22.4. uxstathashindex(regclass) returns record

uxstathashindex返回HASH索引信息的记录。如下所示。

HASH索引的创建，请参见《优炫数据库参考手册 V2.1》中”SQL命令“章节。

```
test=> select * from uxstathashindex('con_hash_index');
-[ RECORD 1 ]--+-----------------
version        | 4
bucket_pages   | 33081
overflow_pages | 0
bitmap_pages   | 1
unused_pages   | 32455
live_items     | 10204006
dead_items     | 0
free_percent   | 61.8005949100872
```

表 22.4. uxstathashindex输出列

列	类型	描述
version	integer	HASH版本号。
bucket_pages	bigint	存储桶页面的数量。
overflow_pages	bigint	溢出页面的数量。
bitmap_pages	bigint	位图页数。
unused_pages	bigint	未使用页面的数量。
live_items	bigint	活元组的数量。
dead_tuples	bigint	死元组的数量。
free_percent	float	自由空间的百分比。

22.5. ux_relpages(regclass) returns bigint

ux_relpages返回关系中的页面数。

22.6. uxstattuple_approx(regclass) returns record

uxstattuple_approx是uxstattuple的一个更加快速的替代品，它返回近似的结果。参数是目标关系的OID或者名称。如下所示。

```
test=> SELECT * FROM uxstattuple_approx('ux_catalog.ux_proc'::regclass);
-[ RECORD 1 ]--------+-------
table_len            | 573440
scanned_percent      | 2
approx_tuple_count   | 2740
approx_tuple_len     | 561210
approx_tuple_percent | 97.87
dead_tuple_count     | 0
dead_tuple_len       | 0
dead_tuple_percent   | 0
approx_free_space    | 11996
approx_free_percent  | 2.09
```

鉴于uxstattuple总是执行全表扫描并且返回存活和死亡元组的准确计数、尺寸和空闲空间，uxstattuple_approx尝试避免全表扫描并且返回死亡元组的准确统计信息，以及存活元组和空闲空间的近似数量及尺寸。

这个函数通过根据可见性映射跳过只包含可见元组的页面来实现这一目的（如果一个页面对应的VM位被设置，那么就说明该页面不含有死亡元组）。对于这样的页面，它会从空闲空间映射中得到空闲空间值，并且假定该页面上的剩余空间由存活元组占据。

对于不能被跳过的页面，它会扫描每个元组，在合适的计数器中记录它的存在以及尺寸，并且统计该页面上的空闲空间。最后，它会基于已扫描的页面和元组数量来估计存活元组的总数（采用与VACUUM估计ux_class.reltuples时相同的方法）。

表 22.5. uxstattuple_approx输出列

列	类型	描述
able_len	bigint	以字节计的物理关系长度（准确）。
scanned_percent	float8	已扫描表的百分比。
approx_tuple_count	bigint	存活元组的数量（估计）。
approx_tuple_len	bigint	以字节计的存活元组总长度（估计）。
approx_tuple_percent	float8	存活元组的百分比。
dead_tuple_count	bigint	死亡元组的数量（准确）。
dead_tuple_len	bigint	以字节计的死亡元组总长度（准确）。
dead_tuple_percent	float8	死亡元组的百分比。
approx_free_space	bigint	以字节计的总空闲空间（估计）。
approx_free_percent	float8	空闲空间的百分比。

在上述的输出中，空闲空间数据可能不完全匹配uxstattuple的输出，这是因为空闲空间映射会给出一个确切的数字，但是这个数字不能保证是准确的字节数。

第 23 章 ux_cron

ux_cron是一个简单的定时任务管理器，可以进行一些例如定期向表中插入数据的操作，它以扩展的方式运行在UXDB数据库上。

23.1. 配置

1. 安装UXDB(version >= 2.1.0.2)，初始化数据库集群。

2. 修改uxsinodb.conf配置文件，添加如下内容。

   ```
   shared_preload_libraries = '' #(change restart required)
   shared_preload_libraries = 'ux_cron'
   cron. database_name = 'uxdb'
   ```

   > **注意**
   >
   > 配置了cron.database_name参数才能在对应的数据库上加载ux_cron扩展，否则创建扩展会报错。

3. 配置ux_hba.conf中密码验证方式为trust，也可以配置.uxpass文件。

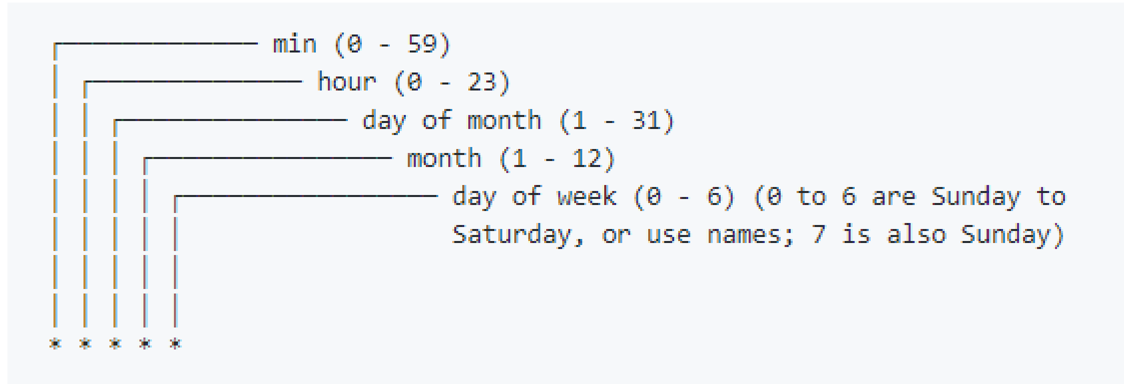
   ```
   #"local" is for Unix domain socket connections only
   local all all trust 00:00-24:00
   #IPv4 local connections:
   host all all 0.0.0.0/0 trust 00:00-24:00
   #IPv6 local connections:
   host all all ::1/128 trust 00:00-24:00
   host replication all localhost trust 00:00-24:00
   ```

4. 启动集群，创建ux_cron扩展。

   ```
   [uxdb@uxdev bin]$ ./uxsql
   uxsql (2.1.0.2)
   Type "help" for help.
   uxdb=# create extension ux_cron;
   CREATE EXTENSION
   ```

23.2. 示例

根据需要进行定时设置。



以每隔1分钟向time_cron表中插入一条当前时间的记录为例。

1. 创建一张time_cron表。

   create table time_cron(data timestamp);

2. 建立定时任务，每分钟向time_cron表中插入一条当前时间记录。

   - 通过函数创建定时任务

     SELECT cron.schedule('*/1 * * * *', 'insert into time_cron values (now())');

   - 通过非函数创建定时任务（ip参数写localhost也可以执行成功）

     INSERT INTO cron.job (schedule, command, nodename, nodeport, database, username) VALUES ('*/1 * * * *', 'insert into time_cron values (now())', '127.0.0.1', 5432, 'uxdb', 'uxdb');

3. 查看任务是否创建成功。

   select * from cron.job;

4. 等待几分钟，查看time_cron表中数据。

   select * from time_cron;

```
uxdb=# create table time_cron (date timestamp);
CREATE TABLE
uxdb=# SELECT cron.schedule('*/1 * * * *', 'insert into time_cron values (now())');
 schedule
----------
        2
(1 row)

uxdb=# select * from cron.job;
 jobid |  schedule   |               command                | nodename  | nodeport | database | username | active
-------+-------------+--------------------------------------+-----------+----------+----------+----------+--------
     2 | */1 * * * * | insert into time_cron values (now()) | localhost |     5432 | uxdb     | uxdb     | t
(1 row)

uxdb=# select * from time_cron;
           date
----------------------------
 2020-04-20 09:38:00.004176
 2020-04-20 09:39:00.004388
 2020-04-20 09:40:00.005153
(3 rows)
```

5. 取消任务。

   - 方式一

     SELECT cron.unschedule(2);

   - 方式二

     delete from cron.job where jobid=2;

第 24 章 ux_hint_plan

24.1. 概述

ux_hint_plan通过向查询提供指示执行计划的提示来控制执行计划。

与Oracle hint功能相似，UADB也有自己的hint，它是以插件形式存在的。它依赖数据库对象的统计信息，统计信息的及时性和准确性都会影响CBO作出最优的决策，在某些需要人为干预的优化决定制定的情况下，hint就发挥了作用。 hint是sql语句级别干预优化器优化过程的方式，它告诉优化器按照我们的告诉它的方式生成执行计划。目前比较主流的数据库Oracle实现了自己的hint，功能如下所示。

1. 使用的优化器的类型；
2. 基于代价的优化器的优化目标，是all_rows还是first_rows；
3. 表的访问路径，是全表扫描，还是索引扫描，还是直接利用rowid；
4. 表之间的连接类型；
5. 表之间的连接顺序；
6. 语句的并行程度等。

UXDB有自己的Hint插件——ux_hint_plan，与Oracle hint有相似的功能。

24.2. 语法

ux_hint_plan语法由/*+ hint[text] */定义，添加在explain关键字之前，具体语法如下。

/*+ hint[text] */ EXPLAIN sqlstament;

如果有多个hint条件，则各个hint[text]中间用空格隔开。

/*+ hint[text] hint[text] … */ EXPLAIN sqlstatment;

其中，“+”号表示该注释是一个hints，该加号必须立即跟在”/*”的后面，且中间不能有空格。

表 24.1. hint提示语句

组	格式	描述
扫描方式	SeqScan(table)	在表上进行顺序扫描
	TidScan(table)	在表上进行TID扫描
	IndexScan(table[index...])	如果有指定的索引，则作为表扫描过程中使用的索引
	IndexOnlyScan(table[index...])	仅适用索引浏览
	BitmapScan(table[index...])	使用特定的索引对表进行位图扫描

组	格式	描述
	NoSeqScan(table)	不对表进行顺序扫描
	NoTidScan(table)	不对表进行TID扫描
	NoIndexScan(table))	不对表进行索引或者特定的索引扫描
	NoIndexOnlyScan(table)	不用索引只扫描表
	NoBitmapScan(table)	不对表进行位图扫描
连接方式	NestLoop(table table[table...])	对包含特定表的连接做嵌套循环
	HashJoin(table table[table...])	对包含特定表的连接做哈希连接
	MergeJoin(table table[table...])	对包含特定表的连接做联合连接
	NoNestLoop(table table[table...])	对包含特定表的连接强制不做嵌套循环
	NoHashJoin(table table[table...])	对包含特定表的连接强制不做哈希连接
	NoMergeJoin(table table[table...])	对包含特定表的连接强制不做联合连接
连接顺序	Leading(table table[table...])	按照特定的连接顺序合并
	Leading(< join pair >)	按照特定的连接顺序和方向连接。连接对是一组表，和/或用括号括起来的可以组成嵌套结构的其他连接对
行号校正	Rows(table table[table...] correction)	更正联接结果中包含特殊表的行号。可用的校正方法有绝对(# n)、加(+ n)、减(- n)和乘(* n)。N 应该是strtod()可以读取的字符串
GUC	Set(GUC-param value)	在计划程序运行时，需要设置GUC参数

24.3. 示例

1. 创建数据库testdb，配置数据库uxsinodb.conf文件。

   vim testdb/uxsinodb.conf

   设置预加载共享库项shared_preload_libraries，并在文件尾添加配置参数。

   ```
   shared_preload_libraries = 'ux_hint_plan'
   ux_hint_plan.enable_hint = on
   ux_hint_plan.debug_print = on
   ux_hint_plan.message_level = log
   ```

2. 配置完成后，启动数据库testdb，连接ux_hint_plan。

create extension ux_hint_plan;
create table a(id int primary key, info text, crt_time timestamp);
create table b(id int primary key, info text, crt_time timestamp);
insert into a select generate_series(1,100000), 'a_'||md5(random()::text), clock_timestamp();
insert into b select generate_series(1,100000), 'b_'||md5(random()::text), clock_timestamp();
analyze a;
analyze b;

结果如图所示。

```
[uxdb@localhost bin]$ uxsql
Password:
uxsql (2.1.0.2)
Type "help" for help.

uxdb=# create extension ux_hint_plan;
CREATE EXTENSION
uxdb=# create table a(id int primary key, info text, crt_time timestamp);
CREATE TABLE
uxdb=# create table b(id int primary key, info text, crt_time timestamp);
CREATE TABLE
uxdb=# insert into a select generate_series(1,100000), 'a_'||md5(random()::text)
, clock_timestamp();
INSERT 0 100000
uxdb=# insert into b select generate_series(1,100000), 'b_'||md5(random()::text)
, clock_timestamp();
INSERT 0 100000
uxdb=# analyze a;
ANALYZE
uxdb=# analyze b;
ANALYZE
uxdb=#
```

3. 使用hint和不使用hint结果对比。

   执行以explain语句，查看结果。

explain select a.*,b.* from a,b where a.id=b.id and a.id < 10;

```
uxdb=# explain select a.*,b.* from a,b where a.id=b.id and a.id<10;
                                QUERY PLAN
--------------------------------------------------------------------------
 Nested Loop  (cost=0.58..83.24 rows=9 width=94)
   ->  Index Scan using a_pkey on a  (cost=0.29..8.45 rows=9 width=47)
         Index Cond: (id < 10)
   ->  Index Scan using b_pkey on b  (cost=0.29..8.31 rows=1 width=47)
         Index Cond: (id = a.id)
(5 rows)

uxdb=#
```

没有使用ux_hint_plan时，需要使用开关来改变UXDB的执行计划，查看结果。

set enable_nestloop=off;
explain select a.*,b.* from a,b where a.id=b.id and a.id < 10;
set enable_nestloop=on;
explain select a.*,b.* from a,b where a.id=b.id and a.id < 10;

```
uxdb=# set enable_nestloop=off;
SET
uxdb=#  explain select a.*,b.* from a,b where a.id=b.id and a.id<10;
                                  QUERY PLAN
------------------------------------------------------------------------------
 Merge Join  (cost=0.65..2274.86 rows=9 width=94)
   Merge Cond: (a.id = b.id)
   ->  Index Scan using a_pkey on a  (cost=0.29..8.45 rows=9 width=47)
         Index Cond: (id < 10)
   ->  Index Scan using b_pkey on b  (cost=0.29..2016.29 rows=100000 width=47)
(5 rows)

uxdb=# set enable_nestloop=on;
SET
uxdb=# explain select a.*,b.* from a,b where a.id=b.id and a.id<10;
                              QUERY PLAN
----------------------------------------------------------------------
 Nested Loop  (cost=0.58..83.24 rows=9 width=94)
   ->  Index Scan using a_pkey on a  (cost=0.29..8.45 rows=9 width=47)
         Index Cond: (id < 10)
   ->  Index Scan using b_pkey on b  (cost=0.29..8.31 rows=1 width=47)
         Index Cond: (id = a.id)
(5 rows)

uxdb=#
```

使用ux_hint_plan来改变Uxdb的执行计划，查看结果。

/*+ HashJoin(a b) SeqScan(b) */ explain select a.*,b.* from a,b where a.id=b.id and a.id < 10;
/*+ SeqScan(a) */ explain select * from a where id < 10;
/*+ BitmapScan(a) */ explain select * from a where id < 10;

```
uxdb=# /*+ HashJoin(a b) SeqScan(b) */ explain select a.*,b.* from a,b where a.i
d=b.id and a.id<10;
                                    QUERY PLAN
----------------------------------------------------------------------------
 Hash Join  (cost=8.56..2504.05 rows=9 width=94)
   Hash Cond: (b.id = a.id)
   ->  Seq Scan on b  (cost=0.00..1233.00 rows=100000 width=47)
   ->  Hash  (cost=8.45..8.45 rows=9 width=47)
         ->  Index Scan using a_pkey on a  (cost=0.29..8.45 rows=9 width=47)
               Index Cond: (id < 10)
(6 rows)

uxdb=# /*+ SeqScan(a) */ explain select * from a where id<10;
                      QUERY PLAN
-----------------------------------------------------
 Seq Scan on a  (cost=0.00..1483.00 rows=9 width=47)
   Filter: (id < 10)
(2 rows)

uxdb=# /*+ BitmapScan(a) */ explain select * from a where id<10;
                              QUERY PLAN
---------------------------------------------------------------------
 Bitmap Heap Scan on a  (cost=4.36..35.17 rows=9 width=47)
   Recheck Cond: (id < 10)
   ->  Bitmap Index Scan on a_pkey  (cost=0.00..4.36 rows=9 width=0)
         Index Cond: (id < 10)
(4 rows)

uxdb=#
```

第 25 章 ux_prewarm

ux_prewarm模块提供一种方便的方法把关系数据载入到操作系统缓冲区或者UXDB缓冲区。

使用ux_prewarm模块前，首先需要执行CREATE EXTENSION命令，如下所示。

```
CREATE EXTENSION ux_prewarm;
ux_prewarm(regclass, mode text default 'buffer', fork text default 'main',
        first_block int8 default null,
        last_block int8 default null) returns int8;
```

第一个参数是要预热的关系。第二个参数是要使用的预热方法，下文将会进一步讨论。第三个参数是要被预热的关系分叉，通常是main。第四个参数是要预热的第一个块号（NULL也被接受，它等同于零）。第五个参数是要预热的最后一个块号（NULL表示一直预热到关系的最后一个块）。返回值是被预热的块数。

有三种可用的预热方法。

1. prefetch会向操作系统发出异步预取请求（如果支持异步预取），不支持异步预取则抛出一个错误。

2. read会读取要求范围的块。与prefetch不同，它是同步的并且在所有平台上都被支持，但是可能较慢。

3. buffer会把要求范围的块读入数据库的缓冲区。

> **注意**
>
> 使用任意一种方法尝试预热比能缓存的数量更多的块—将很可能导致高编号块被读入时把低编号的块从缓冲区中挤出的情况。被预热的数据也不享受对缓冲区替换的特别保护，因此其他系统活动可能会在刚刚被预热的块被读入后很快就将它们逐出。反过来，预热也可能把其他数据逐出缓存。由于这些原因，预热通常在启动时最有用，因为这时缓冲大部分都为空。

第 26 章 ux_stat_statements

ux_stat_statements模块提供一种方法追踪一个服务器已执行的所有SQL语句的执行统计信息。

使用ux_stat_statements模块前，首先需要执行CREATE EXTENSION命令，如下所示。

CREATE EXTENSION ux_stat_statements;

该模块必须通过在uxsinodb.conf的shared_preload_libraries中增加ux_stat_statements来载入，因为它需要额外的共享内存。这意味着增加或移除该模块需要一次服务器重启。

当ux_stat_statements被载入时，它会跟踪该服务器的所有数据库的统计信息。该模块提供了一个视图ux_stat_statements以及函数ux_stat_statements_reset和ux_stat_statements用于访问和操纵这些统计信息。这些视图和函数不是全局可用的，但是可以用CREATE EXTENSION ux_stat_statements为特定数据库启用它们。

26.1. ux_stat_statements视图

由该模块收集的统计信息可以通过一个名为ux_stat_statements的视图使用。这个视图为每一个可区分的数据库ID、用户ID和查询ID（最多到该模块可以追踪的可区分语句的数量）的组合都包含一行。

表 26.1. ux_stat_statements列

列	类型	引用	描述
userid	oid	ux_authid.oid	执行该语句的用户的OID。
dbid	oid	ux_database.oid	在其中执行该语句的数据库的OID。
queryid	bigint		内部哈希码，从语句的解析树计算得来。
query	text		语句的文本形式。
calls	bigint		被执行的次数。
total_time	double precision		在该语句中花费的总时间，以毫秒计。
min_time	double precision		在该语句中花费的最小时间，以毫秒计。
max_time	double precision		在该语句中花费的最大时间，以毫秒计。
mean_time	double precision		在该语句中花费的平均时间，以毫秒计。
stddev_time	double precision		在该语句中花费时间的总体标准偏差，以毫秒计。
rows	bigint		该语句检索或影响的行总数。

列	类型	引用	描述
shared_blks_hit	bigint		该语句造成的共享块缓冲命中总数。
shared_blks_read	bigint		该语句读取的共享块的总数。
shared_blks_dirtied	bigint		该语句造成的脏共享块的总数。
shared_blks_written	bigint		该语句写入的共享块的总数。
local_blks_hit	bigint		该语句造成的本地块缓冲命中总数。
local_blks_read	bigint		该语句读取的本地块的总数。
local_blks_dirtied	bigint		该语句造成的脏本地块的总数。
local_blks_written	bigint		该语句写入的本地块的总数。
temp_blks_read	bigint		该语句读取的临时块的总数。
temp_blks_written	bigint		该语句写入的临时块的总数。
blk_read_time	double precision		该语句花在读取块上的总时间，以毫秒计（如果track_io_timing被启用，否则为零）。
blk_write_time	double precision		该语句花在写入块上的总时间，以毫秒计（如果track_io_timing被启用，否则为零）。

由于安全性原因，只有系统管理员和ux_read_all_stats角色的成员被允许看到其他用户执行的查询的SQL文本或者queryid。不过，如果该视图被安装在其他用户的数据库中，那么他们就能够看见统计信息。

只要可规划的查询（即SELECT、INSERT、UPDATE以及DELETE）根据一种内部哈希计算具有相同的查询结构，它们就会被组合到一个单一的ux_stat_statements项。通常，如果两个查询除了查询中的文本常量值之外在语义上等效，它们将会被认为是相同的。不过，功能性命令（即所有其他命令）会严格地以它们的文本查询字符串为基础进行比较。

当为了把一个查询与其他查询匹配，常数值会被忽略，在ux_stat_statements显示中它会被一个参数符号，比如$1所替换。查询文本的剩余部分就是具有与该ux_stat_statements项相关的特定queryid哈希值的第一个查询的文本。

在某些情况中，具有明显不同文本的查询可能会被融合到一个单一的ux_stat_statements项。通常这只会发生在语义等价的查询身上，但是也有很小的机会因为哈希碰撞的原因导致无关的查询被融合到一个项中（不过，对于属于不同用户或数据库的查询来说不会发生这种情况）。

由于queryid哈希值是根据查询被解析和分析后的表达计算的，对立的情况也可能存在：如果具有相同文本的查询由于参数（如不同的search_path设置）的原因而具有不同的含义，它们就可能作为不同的项存在。

ux_stat_statements的使用者可能希望使用queryid（也许会与dbid和userid组合）作为比查询文本更稳定和可靠的每个条目的标识符。但是，有一点很重要的是，对于queryid哈希值稳定性只有有限的保障。因为该标识符是从post-parse-analysis tree得来的，它的值是以这种形式出现的内部对象标识符的函数。这有一些违背直觉的含义，例如：如果有两个查询引用了同一个表，但是该表在两次查询之间被删除并且重建，显然这两个查询是完全一致的，但是ux_stat_statements将把它们认为是不同的。哈希处理也对机器架构以及平台的其他方面的差别很敏感。更进一步，假定queryid在UXDB的主要版本中都是稳定的是不安全的。

根据经验，只有在底层服务器版本以及目录元数据细节保持完全相同时，queryid值才能被假定为稳定并且可比。两台参与到基于物理WAL重放的复制中的服务器会对相同的查询给出一样的queryid值。但是，逻辑复制模式并不保证在所有相关细节上都保持完全一样的复制，因此在逻辑复制机之间计算代价时，queryid并非是一个有用的标识符。

代表性查询文本中用于替换常量的参数符号从原始查询文本中最高的$n参数之后的下一个数字开始，如果没有则为$1。值得注意的是，在某些情况下，可能存在影响编号的隐藏参数符号。例如，PL/uxSQL使用隐藏参数符号将函数局部变量的值插入到查询中，以便像SELECT i + 1 INTO j的PL/uxSQL语句将具有像SELECT i + $2这样的代表性文本。

有代表性的查询文本被保存在一个外部磁盘文件中，并且不会消耗共享内存。因此，即便是很长的查询文本也能被成功存储。不过，如果累积了很多长查询文本，该外部文件也会增长到很大。作为一种恢复方法，如果这样的情况发生，ux_stat_statements可能会选择丢弃这些查询文本，于是ux_stat_statements视图中的所有现有项将会显示空的query域，不过与每个queryid相关联的统计信息会被保留下来。如果发生这种情况，可以考虑减小ux_stat_statements.max来防止复发。

26.2. 函数

26.2.1. ux_stat_statements_reset() returns void

ux_stat_statements_reset清空目前由ux_stat_statements收集的所有统计信息。默认情况下，这个函数只能被系统管理员执行。

26.2.2. ux_stat_statements(showtext boolean) returns setof record

ux_stat_statements视图按照名为ux_stat_statements的函数来定义。客户端可以直接调用ux_stat_statements函数，并且通过指定showtext := false来忽略查询文本（即，对应于视图的query列的OUT参数将返回空值）。这个特性是为了支持不想重复接收长度不定的查询文本的外部工具而设计的。这类工具可以转而自行缓存第一个查到的查询文本，因为这就是ux_stat_statements所做的全部工作，并且只在需要的时候检索查询文本。因为服务器会把查询文本存储在一个文件中，这种方法可以降低重复检查ux_stat_statements数据的物理I/O。

26.3. 配置参数

26.3.1. ux_stat_statements.max (integer)

ux_stat_statements.max是由该模块跟踪的语句的最大数目（即ux_stat_statements视图中行的最大数量）。如果观测到的可区分的语句超过这个数量，最少被执行的语句的信息将会被丢弃。默认值为5000。这个参数只能在服务器启动时设置。

26.3.2. ux_stat_statements.track (enum)

ux_stat_statements.track控制哪些语句会被该模块计数。指定top可以跟踪顶层语句（那些直接由客户端发出的语句），指定all还可以跟踪嵌套的语句（例如在函数中调用的语句），指定none可以禁用语句统计信息收集。默认值是top。只有系统管理员能够修改该设置。

26.3.3. ux_stat_statements.track_utility (boolean)

ux_stat_statements.track_utility控制该模块是否会跟踪工具命令。工具命令是除了SELECT、INSERT、UPDATE和DELETE之外所有的其他命令。默认值是on。只有系统管理员能够修改该设置。

26.3.4. ux_stat_statements.save (boolean)

ux_stat_statements.save指定是否在服务器关闭之后还保存语句统计信息。如果被设置为off，那么关闭后不保存统计信息并且在服务器启动时也不会重新载入统计信息。默认值为on。这个参数只能在uxsinodb.conf文件中或者在服务器命令行上设置。

该模块要求与ux_stat_statements.max成比例的额外共享内存。注意只要该模块被载入就会消耗这么多的内存，即便ux_stat_statements.track被设置为none。

这些参数必须在uxsinodb.conf中设置。典型的用法可能是：

```
# uxsinodb.conf
shared_preload_libraries = 'ux_stat_statements'
ux_stat_statements.max = 10000
ux_stat_statements.track = all
```

26.4. 示例

```
bench=# SELECT ux_stat_statements_reset();
$ uxbench -i bench
$ uxbench -c10 -t300 bench
bench=# \x
bench=# SELECT query, calls, total_time, rows, 100.0 * shared_blks_hit /
   nullif(shared_blks_hit + shared_blks_read, 0) AS hit_percent
  FROM ux_stat_statements ORDER BY total_time DESC LIMIT 5;
-[ RECORD 1 ]---------------------------------------------------------------------
query       | UPDATE uxbench_branches SET bbalance = bbalance + $1 WHERE bid = $2;
calls       | 3000
total_time  | 9609.00100000002
rows        | 2836
hit_percent | 99.9778970000200936
-[ RECORD 2 ]---------------------------------------------------------------------
query       | UPDATE uxbench_tellers SET tbalance = tbalance + $1 WHERE tid = $2;
calls       | 3000
total_time  | 8015.156
rows        | 2990
hit_percent | 99.9731126579631345
-[ RECORD 3 ]---------------------------------------------------------------------
query       | copy uxbench_accounts from stdin
calls       | 1
```

```
total_time  | 310.624
rows        | 100000
hit_percent | 0.30395136778115501520
-[ RECORD 4 ]------------------------------------------------------------------
query       | UPDATE uxbench_accounts SET abalance = abalance + $1 WHERE aid = $2;
calls       | 3000
total_time  | 271.741999999997
rows        | 3000
hit_percent | 93.7968855088209426
-[ RECORD 5 ]------------------------------------------------------------------
query       | alter table uxbench_accounts add primary key (aid)
calls       | 1
total_time  | 81.42
rows        | 0
hit_percent | 34.4947735191637631
```

第 27 章 xml2

xml2模块提供XPath查询和XSLT功能。

使用xml2模块前，首先需要执行CREATE EXTENSION命令，如下所示。

```
CREATE EXTENSION xml2;
```

27.1. 函数

下表是这个模块提供的函数。这些函数提供了直接的XML解析和XPath查询。所有参数都是text类型，为了简洁都没有被显示。

表 27.1. 函数

函数	返回	描述
xml_valid(document)	bool	这个函数解析其参数中的文档文本，如果该文档是一个结构良好的XML则返回真（注意，这是标准UXDB函数xml_is_well_formed()的别名。xml_valid()的名称在技术上是不正确的，因为在XML中有效性和结构良好性具有不同的含义。）
xpath_string(document, query)	text	这些函数在提供的文档上计算XPath查询，并且将结果转换为指定的类型。
xpath_number(document, query)	float4	
xpath_bool(document, query)	bool	
xpath_nodeset(document, query, toptag, itemtag)	text	这个函数在文档上计算查询并且把结果封装在XML标签中。如果结果是多值的，输出如下示例： <toptag> <itemtag>Value 1 which could be an XML fragment</itemtag> <itemtag>Value 2....</itemtag> </toptag> 如果toptag或者itemtag是一个空字符串，相关的标签会被忽略。
xpath_nodeset(document, query)	text	与xpath_nodeset(document, query, toptag, itemtag)相似但是结果忽略两种标签。
xpath_nodeset(document, query, itemtag)	text	与xpath_nodeset(document, query, toptag, itemtag)相似但是结果忽略toptag。

函数	返回	描述
xpath_list(document, query, separator)	text	这个函数返回多个值，并且用指定的分隔符分隔，例如分隔符是,，结果就是Value 1,Value 2,Value 3。
xpath_list(document, query)	text	这是上面函数的一个封装器，它用,作为分隔符。

27.2. xpath_table

xpath_table(text key, text document, text relation, text xpaths, text criteria) returns setof record

xpath_table是一个表函数，它在一组文档中的每一个上计算一组XPath查询，并且将结果作为一个表返回。来自原始文档表的主键字段被返回为结果的第一列，这样结果集可以被用于连接。其参数如下表所示。

表 27.2. xpath_table参数

参数	描述
key	表的键字段名，用作输出第一列，描述输出结果来源。
document	表中xml字段名，该字段存储的是xml格式的文本。
relation	包含文档的表或视图的名称。
xpaths	xpath表达式，多个表达式用 | 隔开，指定xml文档域。
criteria	WHERE子句的内容。这不能被忽略，因此如果想要处理关系中的所有行，可以使用true或1=1。

这些参数（除了XPath字符串）只是会被替换到一个纯粹的SQL SELECT语句中，因此应该有一些灵活性，如下所示。

SELECT <key>, <document> FROM <relation> WHERE <criteria>

因此那些参数可以是那些特定位置上合法的任何值。SELECT的结果需要返回正好两列（除非尝试为键或文档列出多个域）。注意这种简单方法要求验证任何用户提供的值，避免SQL注入攻击。

示例

1. 加载xml2扩展插件。

```
create extension xml2;
```

2. 建立测试表，包含id和documents两个字段，其中documents字段类型为xml。

```
CREATE TABLE xmldata (id bigserial not null, documents xml);
insert into xmldata values(1, '<test color="red">Red</test>');
insert into xmldata values(2, '<test color="green">Green</test>');
uxdb=# select * from xmldata;
 id |           documents
----+-----------------------------------
```

```
 1 | <test color="red">Red</test>
 2 | <test color="green">Green</test>
(2 rows)
```

3. 通过xpath_table函数从表中检索关键字。

```
uxdb=# SELECT * FROM xpath_table('id','documents', 'xmldata', '/test', 'true')
 AS t(doc_id integer, data text);
 doc_id |  data
--------+----------
      1 | Red
      2 | Green
```

该函数必须被使用在一个FROM表达式中，并带有一个AS子句来指定输出列，例如：

```
SELECT * FROM
xpath_table('article_id',
            'article_xml',
            'articles',
            '/article/author|/article/pages|/article/title',
            'date_entered > ''2003-01-01'' ')
AS t(article_id integer, author text, page_count integer, title text);
```

AS子句定义了输出表中列的名称和类型。第一个是“key”域并且剩下的对应于XPath查询。如果XPath查询比结果列多，额外的查询将被忽略。如果结果列比XPath查询多，额外的列将是NULL。

> **注意**
>
> 这个示例定义page_count结果列为一个整数。该函数在内部处理字符串表达，因此在输出中想要一个整数时，它将采用XPath结果的字符串表达并且使用UXDB输入函数来将其转换成一个整数（或者AS子句要求的任何类型）。如果它无法做到这一点将会导致一个错误—例如结果是空—因此如果数据有任何问题，可能希望坚持用text作为列类型。

调用的SELECT语句不必只是SELECT * — 它可以用名称引用输出列或者将它们连接到其他表。该函数会产生一个虚拟表，可以在其上执行任何所需的操作（例如聚集、连接、排序等）。因此更复杂的示例如下所示。

```
SELECT t.title, p.fullname, p.email
  FROM xpath_table('article_id', 'article_xml', 'articles',
               '/article/title|/article/author/@id',
               'xpath_string(article_xml,''/article/@date'') > ''2003-03-20'' ')
    AS t(article_id integer, title text, author_id integer),
     tblPeopleInfo AS p
 WHERE t.author_id = p.person_id;
```

当然，为了便利也可以把所有这些封装在一个视图中。

27.2.1. 多值结果

xpath_table函数假定每一个XPath　查询的结果可能是多值的，因此该函数返回的行数可能与输入文档的数目不同。被返回的第一行包含来自每一个查询的第一个结果，第二行则是来自每一个查询的第二个结果。如果其中一个查询的值比其他查询少，则会为它返回空值。

在某些情况下，一个用户将知道一个给定的XPath 查询将只返回一个单一结果（可能是一个唯一文档标识符） — 如果和一个返回多值的XPath 查询一起使用，单值结果将只出现在结果的第一行中。对于这种情况的解决方案是使用键域作为针对一个更简单 XPath查询的连接的一部分。如下所示。

```
CREATE TABLE test (id int PRIMARY KEY, xml text);
INSERT INTO test VALUES (1, '<doc num="C1">
<line num="L1"><a>1</a><b>2</b><c>3</c></line>
<line num="L2"><a>11</a><b>22</b><c>33</c></line>
</doc>');
INSERT INTO test VALUES (2, '<doc num="C2">
<line num="L1"><a>111</a><b>222</b><c>333</c></line>
<line num="L2"><a>111</a><b>222</b><c>333</c></line>
</doc>');
SELECT * FROM
  xpath_table('id','xml','test',
          '/doc/@num|/doc/line/@num|/doc/line/a|/doc/line/b|/doc/line/c',
          'true')
    AS t(id int, doc_num varchar(10), line_num varchar(10), val1 int, val2 int, val3 int)
 WHERE id = 1 ORDER BY doc_num, line_num;
 id | doc_num | line_num | val1 | val2 | val3
----+---------+----------+------+------+------
  1 | C1      | L1       |    1 |    2 |    3
  1 |         | L2       |   11 |   22 |   33
```

要在每一行上得到doc_num，解决方案是使用xpath_table的两个调用并且连接结果。

```
SELECT t.*,i.doc_num FROM
  xpath_table('id', 'xml', 'test',
          '/doc/line/@num|/doc/line/a|/doc/line/b|/doc/line/c',
          'true')
    AS t(id int, line_num varchar(10), val1 int, val2 int, val3 int),
  xpath_table('id', 'xml', 'test', '/doc/@num', 'true')
    AS i(id int, doc_num varchar(10))
 WHERE i.id=t.id AND i.id=1
 ORDER BY doc_num, line_num;
 id | line_num | val1 | val2 | val3 | doc_num
----+----------+------+------+------+---------
  1 | L1       |    1 |    2 |    3 | C1
  1 | L2       |   11 |   22 |   33 | C1
(2 rows)
```

27.3. XSLT函数

如果安装了libxslt，那么可以使用下面的函数。

xslt_process(text document, text stylesheet, text paramlist) returns text

这个函数将XSL样式表应用于文档并且返回转换过的结果。paramlist是一个被用在转换中的参数赋值列表，以a=1,b=2的形式指定。注意参数解析非常简单：参数值不能包含逗号！

xslt_process还有一个两个参数的版本，它不向转换传递任何参数。