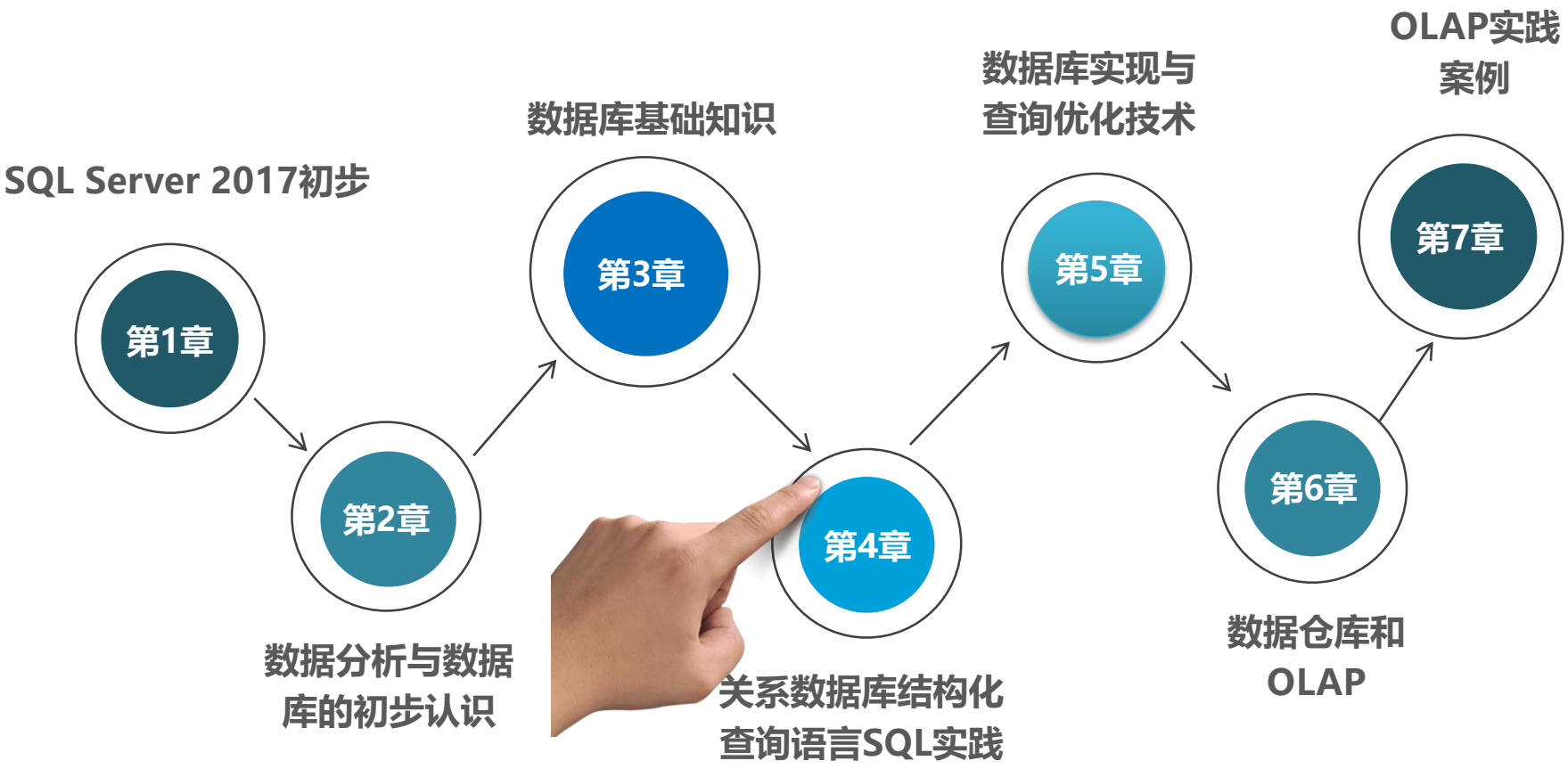
SQL Server 2017

数据库分析处理技术

张延松

中国人民大学 信息学院

目录


SQL Server 2017初步
第1章
第2章
数据分析与数据库的初步认识
数据库基础知识
第3章
第4章
关系数据库结构化查询语言SQL实践
数据库实现与查询优化技术
第5章
第6章
数据仓库和OLAP
OLAP实践案例
第7章

本章内容

本章要点/学习目标

SQL是结构化查询语言（Structured Query Language） 的简称，是关系数据库的标准语言。SQL是一种通用的、功能强大的数据库查询和程序设计语言，用于存取数据以及查询、更新和管理关系数据库系统，几乎所有关系数据库系统都支持SQL，而且一些非关系数据库也支持类似SQL或与SQL部分兼容的查询语言，如Hive SQL、SciDB AQL、SparkSQL等。SQL同样得到其他领域的重视和采用，如人工智能领域的数据检索。同时，SQL语言也在不断发展，SQL标准中增加了对JSON的支持，SQL Server 2017增加了对图数据处理的支持。

本章学习的目标是掌握SQL语言的基本语法与使用技术，能够面向企业级数据库进行管理和数据处理，实现基于SQL的数据分析处理。

第4节 数据更新SQL

- 数据更新的操作包含对表中记录的增、删、改操作，对应的SQL命令分别为INSERT、DELETE和UPDATE。

一、插入数据

- SQL命令中插入语句包括两种类型：插入一个新元组，插入查询结果。插入查询结果时可以一次插入多条元组。

1.插入元组

- 命令：INSERT INTO VALUES
- 功能：在表中插入记录。
- 语法：

```
INSERT INTO <table_name> [column_list]
VALUES ({DEFAULT | NULL | expression } [ ,...n ]);
```

- SQL命令描述：
- INSERT语句的功能是向指定的表table_name中插入元组，column_list指出插入元组对应的属性，可以与表中列的顺序不一致，没有出现的属性赋空值，需要保证没有出现的属性不存在NOT NULL约束，不然会出错。当不使用column_list时需要插入全部的属性值。VALUES子句按column_list顺序为表记录各个属性赋值。

第4节 数据更新SQL

- SQL命令示例：

【例4-48】在REGION表中插入新记录NORTH AMERICA和SOURTH AMERICA。

```
insert into REGION(R_REGIONKEY, R_NAME)
values (5,'NORTH AMERICA');
insert into REGION
values(6,'SOURCE AMERICA',null);
```

	R_REGIONKEY	R_NAME	R_COMMENT
▶	0	AFRICA	special Tiresias about the furiously even dolphins are furi
	1	AMERICA	even, ironic theodolites according to the bold platelets wa
	2	ASIA	silent, bold requests sleep slyly across the quickly sly dependencies. furiously silent instructions alongside
	3	EUROPE	special, bold deposits haggle foxes. platelet
	4	MIDDLE EAST	furiously unusual packages use carefully above the unusual, exp
	5	NORTH AMERICA	*NULL*
	6	SOURCE AMERICA	*NULL*

图4-29 插入操作结果

- SQL命令解析：在REGION表中插入一个新记录，对指定的列R_REGIONKEY、R_NAME分别赋值5,' NORTH AMERICA '，其余未指定列自动赋空值NULL。
- 在INSERT命令中需要保证VALUES子句中值的顺序与INTO子句中列的顺序相对应，第二条插入命令未指定列顺序时，需要按表定义的列顺序输入完整的VALUES值，R_COMMENT列不能缺失，可以输入空值。

2.插入子查询结果

- 语法：

  INSERT INTO <table_name> [column_list]
  SELECT…FROM…;

- SQL命令描述：
  - 将子查询的结果批量地插入表中。要求预先建立记录插入的目标表，然后通过子查询选择记录，批量插入目标表，子查询的列与目标表的列相对应。

【例4-49】将查询结果插入新表中，然后通过ROW_NUMBER函数对以C_CUSTKEY分组统计的订单数量按大小排列和C_CUSTKEY排序并分配行号。

```
create table custkey_counter(CUSTKEY int, counter int);

insert into custkey_counter
select C_CUSTKEY, count (*)
from ORDERS, CUSTOMER
where O_CUSTKEY=C_CUSTKEY
group by C_CUSTKEY;

select CUSTKEY, counter, ROW_NUMBER() over (order by counter) as rownum
from custkey_counter
order by counter, CUSTKEY;
```

第4节 数据更新SQL

- SQL命令解析：首先建立目标表custkey_counter，包含两个int型列。然后通过子查询select C_CUSTKEY, count (*) from ORDERS, CUSTOMER where O_CUSTKEY = C_CUSTKEYgroup by C_CUSTKEY;产生查询结果集，通过INSERT INTO语句将子查询的结果集插入目标表custkey_counter中。最后在custkey_counter上执行分配序号操作。
- SELECT…INTO new_table 也提供了类似的将子查询结果插入目标表的功能。
- SELECT…INTO new_table 命令不需要预先建立目标表，查询根据选择列表中的列和从数据源选择的行，在指定的新表中插入记录。

【例4-50】将上例分组聚集结果插入表custkey_counter1中。

```
select C_CUSTKEY, count(*) as counter into custkey_counter1
from ORDERS, CUSTOMER
where O_CUSTKEY=C_CUSTKEY
group by C_CUSTKEY;
```

- SQL命令解析：因为查询中包含聚集表达式，因此需要为聚集结果列赋一个别名AS counter，作为目标表中的列名。系统自动创建表custkey_counter1，表中包含与C_CUSTKEY和counter一致的列。

第4节 数据更新SQL

二、修改数据

- 修改操作又称为更新操作。
- 命令：UPDATE
- 功能：修改表中元组的值。
- 语法：

  UPDATE <table_name>

  SET <column_name>=<expression>[<column_name>=<expression>]

  [FROM { <table_source> } [,...n]]

  [WHERE <search_condition>];

- SQL命令描述：

  UPDATE语句的功能是更新表中满足WHERE子句条件的记录中由SET指定的属性值。

第4节 数据更新SQL

- SQL命令示例：修改单个记录属性值

【例4-51】将REGION表中R_NAME为NORTH AMERICA记录的C_COMMENT属性设置为including Canada and USA。

```
update REGION set R_COMMENT='including Canada and USA'
where R_NAME='NORTH AMERICA';
```

- SQL命令解析：修改指定单个记录属性值时，WHERE条件通常使用码属性上的等值条件来确定到指定的记录。

第4节 数据更新SQL

- SQL命令示例：按条件修改多个记录属性值

【例4-52】将LINEITEM表中订单平均L_COMMITDATE与L_SHIPDATE间隔时间超过60天的订单的O_ORDERPRIORITY更改为1-URGENT。

```
update ORDERS set O_ORDERPRIORITY ='1-URGENT'
from
(select L_ORDERKEY,
avg (DATEDIFF(DAY, L_COMMITDATE,L_SHIPDATE)) as avg_delay
from LINEITEM
group by L_ORDERKEY
having avg(DATEDIFF(DAY, L_COMMITDATE,L_SHIPDATE))>60) as order_delay
where L_ORDERKEY=O_ORDERKEY;
```

- SQL命令解析：更新ORDERS表中O_ORDERPRIORITY列的值，但更新的条件需要通过连接子查询构造。查询的关键是通过派生表计算出LINEITEM表中按订单号分组计算L_COMMITDATE与L_SHIPDATE平均间隔时间，并给派生表命名，然后派生表与ORDERS表按订单号连接并完成基于连接表的更新操作。

第4节 数据更新SQL

- SQL命令示例：通过子查询修改记录属性值

【例4-53】将INDONESIA国家的供应商的S_ACCTBAL值增加5%。

```
update SUPPLIER set S_ACCTBAL=S_ACCTBAL*1.05
where S_NATIONKEY in (
select N_NATIONKEY from NATION where N_NAME='INDONESIA');
```

- SQL命令解析：通过in嵌套查询将NATION表上的条件传递给SUPPLIER表作为更新条件。

三、删除数据

- 删除操作用于将表中满足条件的记录删除。
- 命令：DELETE
- 功能：删除表中元组。
- 语法：

```
DELETE
FROM <table_name>
[WHERE <search_condition>];
```

- SQL命令描述：

  DELETE语句的功能是删除表中的记录，当不指定WHERE条件时删除表中全部记录，指定WHERE条件时按条件删除记录。

第4节 数据更新SQL

- SQL命令示例：删除表中指定条件的元组

【例4-54】删除custkey_counter1表中全部记录。

```
delete from custkey_counter1;
```

- SQL命令解析：删除指定表中全部记录，如果要删除表，使用DROP TABLE custkey_counter1命令。
- SQL命令示例：删除表中指定条件的元组

【例4-55】删除REGION表中R_NAME为SOURTH AMERICA的记录。

```
delete from REGION WHERE R_NAME='SOURTH AMERICA';
```

– SQL命令解析：按谓词条件删除表中指定的一条或多条记录。

- SQL命令示例：通过子查询删除元组

【例4-56】删除LINEITEM表中订单O_ORDERSTATUS状态为F的记录。

```
delete from LINEITEM
where L_ORDERKEY in (
select O_ORDERKEY from ORDERS where O_ORDERSTATUS='F');
```

- 在具有参照完整性约束关系的表中，删除被参照表记录之前要先删除参照表中对应的记录，然后才能删除被参照表中的记录，实现cascade级联删除，满足约束条件。

四、事务

- 数据库中的事务是用户定义的一个SQL操作序列，事务中的操作序列满足要么全做，要么全不做的要求，是一个用户定义的不可分割的操作单位。SQL定义事务的语句有：

  begin transaction [<transaction name>]

  commit transaction [<transaction name>]

  rollback [<transaction name>]

- 事务以begin transaction为开始，commit表示事务成功提交，rollback表示事务中的操作全部撤销，回滚到事务开始的状态。
- 事务需要满足4个特性：原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）、持久性（Durability），简称ACID特性，数据库保证ACID特性的主要技术是并发控制和恢复机制。

第4节 数据更新SQL

- SQL命令示例：事务控制

【例4-57】在LINEITEM表中插入一条L_PARTKEY为6、L_SUPPKEY为7507的订单记录，L_QUANTITY为100，同时将PARTSUPP表中对应记录的PS_AVAILQTY值减100。SQL命令如下所示：

```
begin transaction orderitem
update PARTSUPP set PS_AVAILQTY=PS_AVAILQTY-100
where PS_PARTKEY=6 and PS_SUPPKEY=7507;
insert into LINEITEM (L_ORDERKEY, L_LINENUMBER, L_PARTKEY,
L_SUPPKEY, L_QUANTITY) values(578,3,6,7507,100);
commit transaction
```

- SQL解析：查询对应两个表上的更新命令，需要保证两个SQL命令序列包含在一个事务中，执行全部的更新命令，或者在出现故障时部分执行的更新命令恢复到初始状态。

本章内容

本章要点/学习目标

SQL是结构化查询语言（Structured Query Language） 的简称，是关系数据库的标准语言。SQL是一种通用的、功能强大的数据库查询和程序设计语言，用于存取数据以及查询、更新和管理关系数据库系统，几乎所有关系数据库系统都支持SQL，而且一些非关系数据库也支持类似SQL或与SQL部分兼容的查询语言，如Hive SQL、SciDB AQL、SparkSQL等。SQL同样得到其他领域的重视和采用，如人工智能领域的数据检索。同时，SQL语言也在不断发展，SQL标准中增加了对JSON的支持，SQL Server 2017增加了对图数据处理的支持。

本章学习的目标是掌握SQL语言的基本语法与使用技术，能够面向企业级数据库进行管理和数据处理，实现基于SQL的数据分析处理。

第5节 视图的定义和使用

- 视图是数据库从一个或多个基本表导出的虚表，视图中只存储视图的定义，但不存放视图对应的实际数据。
- 当访问视图时，通过视图的定义实时地从基本表中读取数据。定义视图为用户提供了基本表上多样化的数据子集，但不会产生数据冗余以及不同数据复本导致的数据不一致问题。
- 视图在定义后可以和基本表一样被查询、删除，也可以在视图上定义新的视图。
- 由于视图并不实际存储数据，视图的更新操作有一定的限制。

第5节 视图的定义和使用

一、定义视图

1.创建视图

- 命令：CREATE VIEW
- 功能：创建一个视图。
- 语法：

  CREATE VIEW <view_name> [(column_name [,...n])]

  AS <select_statement>

  [WITH CHECK OPTION];

- SQL命令描述：
  - 子查询select_statement可以是任意的SELECT语句，WITH CHECK OPTION表示对视图进行UPDATE、INSERT和DELETE操作时要保证更新、插入或删除的行满足视图定义中的谓词条件，即子查询中的条件表达式。
  - 在视图定义时，视图属性列名省略默认视图由子查询中SELECT子句目标列中的各字段组成；当子查询的目标列是聚集函数或表达式、多表连接中同名列或者使用新的列名时需要指定组成视图的所有列名。

第5节 视图的定义和使用

- SQL命令示例：

【例4-58】创建视图revenue，定义1996年1月1日起的3个月内按供应商号对LINEITEM表的折扣后价格进行分组聚集计算，并查询贡献了最高销售额的供应商。

```
create view revenue (SUPPLIER_NO, TOTAL_REVENUE) as
select L_SUPPKEY, sum(L_EXTENDEDPRICE*(1-L_DISCOUNT))
from LINEITEM
where L_SHIPDATE>='1996-01-01' and L_SHIPDATE<DATEADD(MONTH,3,'1996-01-01' )
group by L_SUPPKEY;

select S_SUPPKEY, S_NAME, S_ADDRESS, S_PHONE, TOTAL_REVENUE
from SUPPLIER, REVENUE
where S_SUPPKEY=SUPPLIER_NO and TOTAL_REVENUE = (
select max(TOTAL_REVENUE)
from REVENUE )
order by S_SUPPKEY;

drop view REVENUE;
```

- SQL命令解析：首先定义临时视图revenue在指定的时间段内对销售额按供应商号进行汇总计算，然后将视图与SUPPLIER表进行连接，查询最大销售额对应的供应商信息，查询完成后删除视图。

第5节 视图的定义和使用

- 本例中视图起到临时表的作用，也可以使用WITH表达式完成该查询任务。

```
WITH revenue (supplier_no, total_revenue) as (
select l_suppkey, sum(l_extendedprice*(1-l_discount))
from lineitem
where l_shipdate>='1996-01-01' and l_shipdate<DATEADD(MONTH,3,'1996-01-01' )
group by l_suppkey)
select s_suppkey, s_name, s_address, s_phone, total_revenue
from supplier, revenue
where s_suppkey=supplier_no and total_revenue = (
select max(total_revenue)
from revenue )
order by s_suppkey;
```

- SQL命令解析：WITH表达式定义与视图类似，定义后直接使用表达式进行查询。

第5节 视图的定义和使用

- SQL命令示例：定义多表连接视图

【例4-59】创建TPC-H多表连接视图。

```
create view TPCH_view as
select *
from
PART,SUPPLIER,PARTSUPP,LINEITEM,ORDERS,CUSTOMER,NATION,REGI
ON
where S_SUPPKEY = L_SUPPKEY
and PS_SUPPKEY = L_SUPPKEY
and PS_PARTKEY = L_PARTKEY
and P_PARTKEY = L_PARTKEY
and O_ORDERKEY = L_ORDERKEY
and C_CUSTKEY=O_CUSTKEY
and S_NATIONKEY = N_NATIONKEY
and C_NATIONKEY=N_NATIONKEY
and N_REGIONKEY=R_REGIONKEY;
```

```
drop view if exists TPCH_view;
create view TPCH_view as
select count(*)
from PARTSUPP,LINEITEM,ORDERS,CUSTOMER,PART,SUPPLIER,NATION,REGION
where PS_PARTKEY = L_PARTKEY
and PS_SUPPKEY=L_SUPPKEY
and O_ORDERKEY = L_ORDERKEY
and C_CUSTKEY=O_CUSTKEY
and P_PARTKEY = PS_PARTKEY
and S_SUPPKEY = PS_SUPPKEY
and S_NATIONKEY = N_NATIONKEY
--and C_NATIONKEY=N_NATIONKEY
and N_REGIONKEY=R_REGIONKEY
;

select count(*) from TPCH_view;
select count(*) from LINEITEM;
```

- SQL命令解析：根据TPC-H模式创建全部表间连接视图。

2.删除视图

- 命令：DROP VIEW
- 功能：删除指定的视图。
- 语法：

  DROP VIEW <view_name>;

- SQL命令描述：
  - 删除指定的视图。视图定义在一个或多个基本表上，当视图依赖的基本表被删除时，数据库并不自动删除依赖基本表的视图，但视图已失效，需要通过视图删除命令手式删除失效的视图。
  - 当视图依赖的基本表结构发生改变时，可以通过修改视图的定义维持视图不变，从而为用户提供一个统一的视图访问，消除因数据库结构变化而导致的用户应用失效。
- 删除视图示例如【4-58】所示。

  drop view REVENUE;

第5节 视图的定义和使用

二、查询视图

- 在视图上可以执行与基本表一样的查询操作。
- 数据库执行对视图的查询时，把视图定义的子查询和用户查询结合起来，转换成等价的对基本表的查询。
- SQL命令示例：

【例4-60】在TPC-H多表连接视图上查询。

```
select count(*) from TPCH_view
where P_CONTAINER in ('WRAP BOX','MED CASE','JUMBO PACK');
```

- --SQL解析：根据视图定义与视图上的查询转换成下面等价的SQL命令：

```
select count(*)
from PART,SUPPLIER,PARTSUPP,LINEITEM,ORDERS,CUSTOMER,NATION,REGION
where S_SUPPKEY = L_SUPPKEY
and PS_SUPPKEY = L_SUPPKEY
and PS_PARTKEY = L_PARTKEY
and P_PARTKEY = L_PARTKEY
and O_ORDERKEY = L_ORDERKEY
and C_CUSTKEY=O_ORDERKEY
and S_NATIONKEY = N_NATIONKEY
and C_NATIONKEY=N_NATIONKEY
and N_REGIONKEY=R_REGIONKEY
and P_CONTAINER in ('WRAP BOX','MED CASE','JUMBO PACK');
```

第5节 视图的定义和使用

- 根据查询的语义，其最优的等价查询命令为：

```
select count(*)
from PART,LINEITEM
where P_PARTKEY = L_PARTKEY
and P_CONTAINER in ('WRAP BOX','MED CASE','JUMBO PACK');
```

- 对复杂视图上查询的等价转换取决于数据库查询处理引擎优化器的设计。

三、更新视图

- 更新视图是通过视图执行插入、删除、修改数据的操作。不同于基本表，视图更新有很多限制条件。

1.单表上的视图更新

【例4-61】分析下面视图上支持的更新操作。

```
create view order_vital_items as
select O_ORDERKEY, O_ORDERSTATUS, O_TOTALPRICE,
O_ORDERDATE, O_ORDERPRIORITY
from ORDERS
where O_ORDERPRIORITY in ('1-URGENT','2-HIGH');
```

- SQL解析：视图基于单表选择、投影操作而创建，在视图上执行更新操作时需要根据基本表的定义和约束条件确定更新操作的可行性。

第5节 视图的定义和使用

- 当执行插入（insert）操作时，由于基本表上存在非视图定义属性可能会拒绝插入或者非视图属性设置为空值。
- 当视图属性未包含基本表上的主码时，同样不能完成插入操作。如下面SQL命令通过视图order_vital_items在ORDERS表中插入一条记录，记录中未包含在视图插入命令的列设置为空值，当插入记录values列表中未包含O_ORDERKEY属性值时，由于在基本表上违反了主码约束条件而被拒绝插入。
- 插入命令中O_ORDERPRIORITY属性值不满足视图定义时的条件O_ORDERPRIORITY in ('1-URGENT','2-HIGH')，记录可以插入但在视图中查看不到，可以在基本表上查询到该记录。

```
insert into order_vital_items values(8,'F',23453,'1998-03-23','3-MEDIUM');
```

第5节 视图的定义和使用

- 视图上的删除（delete）与修改（update）操作需要满足视图对应的基本表上的约束条件，如主码唯一或与参照表之间的主码-外码参照关系，若违反则该更新操作被拒绝。当满足执行条件时，视图上的更新操作与视图定义相结合执行。

  update order_vital_items set O_ORDERPRIORITY='3-MEDIUM'

  where O_ORDERDATE='1994-07-10';

- 可以改写为等价的SQL命令：

  update ORDERS set O_ORDERPRIORITY='3-MEDIUM'

  where O_ORDERDATE='1994-07-10' and O_ORDERPRIORITY in ('1-URGENT','2-HIGH');

2.单表上的聚集视图更新

【例4-62】分析下面视图上支持的更新操作。

- 视图ORDERPRIORITY_count为ORDERS表上分组聚集结果集。

```
create view ORDERPRIORITY_count as
select O_ORDERPRIORITY, count(*) as counter
from ORDERS
group by O_ORDERPRIORITY;
```

- 视图对应的不是基本表上的基本数据，而是基本表的计算结果，因此对视图中的记录更新无法转换为等价的SQL命令，被数据库系统拒绝。如：

```
insert into ORDERPRIORITY_count values('6-VERY LOW',50678);
update ORDERPRIORITY_count set counter=50678
where O_ORDERPRIORITY='1-URGENT';
delete from ORDERPRIORITY_count where O_ORDERPRIORITY='1-URGENT';
```

第5节 视图的定义和使用

- **3.多表连接视图更新**
- 【例4-63】分析下面视图上支持的更新操作。
- 创建NATION与REGION基本表的连接视图nation_region。

```
create view nation_region as
select * from NATION, REGION where N_REGIONKEY=R_REGIONKEY;
```

- 插入操作被拒绝，因为视图中的记录来自两个基本表，无法满足基本表上的主码-外码参照关系。

```
insert into nation_region values(25,'USA',1,1,'AMERICA');
```

- 删除操作被拒绝，视图对应的NATION表与REGION表上存在主码-外码参照关系，不允许通过视图删除记录。

```
delete from nation_region where R_NAME='ASIA';
delete from nation_region where N_NAME='ALGERIA';
```

- 修改操作可执行。第一条update命令修改视图中NATION表属性，转换为在NATION表上的update命令update NATION set N_NAME='ALG' where N_NAME='ALGERIA';

```
update nation_region set N_NAME='ALG' where N_NAME='ALGERIA';
```

- 第二条update命令修改视图中REGION表属性，等价的SQL命令为update REGION set R_NAME='AFR' where R_NAME='AFRICA';更新后视图中显示多条记录相关列被更新，实际对应REGION表中一条记录更新。

```
update nation_region set R_NAME='AFR' where R_NAME='AFRICA';
```

第5节 视图的定义和使用

- 不同数据库对视图更新的支持不同，通常支持下列条件视图上的更新：
  - From子句中只有一个基本表
  - Select子句只包含基本表的属性，不包含任何表达式、聚集表达式或distinct声明
  - 任何没有出现在select子句中的属性都可以取空值
  - 定义视图的查询中不包含group by和having子句
- 通常多表连接视图及嵌套子查询视图不支持更新，具体情况还需要参照不同数据库的系统设计。
- 视图可以看作是数据库对外的数据访问接口，它可以屏蔽基本表上的敏感信息，为不同用户定制不同的数据访问视图，简化查询处理，并且能够通过视图屏蔽数据库底层的数据结构变化。

本章内容

本章要点/学习目标

SQL是结构化查询语言（Structured Query Language） 的简称，是关系数据库的标准语言。SQL是一种通用的、功能强大的数据库查询和程序设计语言，用于存取数据以及查询、更新和管理关系数据库系统，几乎所有关系数据库系统都支持SQL，而且一些非关系数据库也支持类似SQL或与SQL部分兼容的查询语言，如Hive SQL、SciDB AQL、SparkSQL等。SQL同样得到其他领域的重视和采用，如人工智能领域的数据检索。同时，SQL语言也在不断发展，SQL标准中增加了对JSON的支持，SQL Server 2017增加了对图数据处理的支持。

本章学习的目标是掌握SQL语言的基本语法与使用技术，能够面向企业级数据库进行管理和数据处理，实现基于SQL的数据分析处理。

第6节 面向大数据管理的SQL扩展语法

- 在大数据分析的批量数据处理、交互式查询、实时流处理三大类型中，交互式查询是一个重要的环节，需要满足用户的ad-hoc即席查询、报表查询、迭代处理等查询需求，需要为用户提供SQL接口来兼容原有数据库用户的工作习惯，便于数据库用户及业务平滑地迁移到大数据分析平台。当前大数据管理平台中一个重要的方面是SQL on Hadoop，通过Hadoop大数据平台扩展SQL的分布式查询处理能力。
- 同时，SQL标准扩展了对非结构化数据类型的支持，如支持JSON数据管理以及图数据处理，本节介绍面向大数据管理领域的SQL扩展语法

第6节 面向大数据管理的SQL扩展语法

中国人民大學 RENMIN UNIVERSITY OF CHINA

一、HiveQL

1. HiveQL创建表命令

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] table_name
  [(col_name data_type [COMMENT col_comment], ...)]
  [COMMENT table_comment]
  [PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]
  [CLUSTERED BY (col_name, col_name, ...)
  [SORTED BY (col_name [ASC|DESC], ...)] INTO num_buckets BUCKETS]
  [ROW FORMAT row_format]
  [STORED AS file_format]
  [LOCATION hdfs_path]
```

- 与SQL的建表命令语法结构类似，其中：
  - EXTERNAL 关键字可以让用户创建一个外部表，在建表的同时指定一个指向实际数据的路径（LOCATION）
  - PARTITIONED BY关键字指定表中用于分区的属性列表，需要指定属性名与数据类型
  - ROW FORMAT关键字指定数据格式
  - STORED AS关键字指定存储文件类型，如TEXTFILE、SEQUENCEFILE、RCFILE、和BINARY SEQUENCEFILE
  - LOCATION关键字指定在分布式文件系统中用于存储数据文件的位置

第6节 面向大数据管理的SQL扩展语法

- Hive采用HDFS分布式存储，在创建表时带有分布式存储的特点。

【例4-64】创建外部表part。

```
create external table part (P_PARTKEY INT, P_NAME STRING, P_MFGR
STRING,
P_BRAND STRING, P_TYPE STRING, P_SIZE INT, P_CONTAINER STRING,
P_RETAILPRICE DOUBLE, P_COMMENT STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'
STORED AS TEXTFILE
LOCATION '/tpch/part';
```

- HiveQL命令解析：创建外部表part，存储为文本文件，列分隔符为“|”，存储位置为“/tpch/part”。与SQL命令相比，外部表需要指定记录行格式、文件存储类型及位置。

2. HiveQL数据加载命令

- Hive不支持使用insert命令逐条插入，也不支持update命令，数据以load的方式批量加载到创建的表中。

【例 4-65】加载数据。

```
LOAD DATA LOCAL INPATH './share/tpch_data/part.tbl' OVERWRITE INTO TABLE part;
```

3. HiveQL查询命令

- HiveQL的基本语法格式如下：

```
SELECT [ALL | DISTINCT] select_expr, select_expr, ...
FROM table_reference
[WHERE where_condition]
[GROUP BY col_list]
[HAVING having_condition]
[CLUSTER BY col_list | [DISTRIBUTE BY col_list] [SORT BY| ORDER BY col_list]]
[LIMIT number];
```

- HiveQL数据查询语法类似SQL，其中ORDER BY 对应全局排序，只有一个Reduce任务，SORT BY 只在本机做排序。

第6节 面向大数据管理的SQL扩展语法

- HiveQL不支持等值连接。SQL中两表连接可以写为select * from R, S where R.a=S.a;在HiveQL中需要写成select * from R join S on R.a=S.a;如下面的SQL与HiveQL查询示例：

【例4-66】SQL与HiveQL连接示例。

- SQL连接示例：

```
select l_orderkey, sum(l_extendedprice*(1-l_discount)) as revenue,
o_orderdate, o_shippriority
from customer, orders, lineitem
where c_mktsegment = ' BUILDING ' and c_custkey = o_custkey
and l_orderkey = o_orderkey and o_orderdate < '1995-03-15' and l_shipdate >
'1995-03-15'
group by l_orderkey, o_orderdate, o_shippriority
order by revenue desc, o_orderdate;
```

第6节 面向大数据管理的SQL扩展语法

- HiveQL连接示例：

```
Select l_orderkey, sum(l_extendedprice*(1-l_discount)) as revenue,
o_orderdate, o_shippriority
from customer c join orders o on c.c_mktsegment = 'BUILDING'
and c.c_custkey = o.o_custkey
join lineitem l on l.l_orderkey = o.o_orderkey
where o_orderdate < '1995-03-15' and l_shipdate > '1995-03-15'
group by l_orderkey, o_orderdate, o_shippriority
order by revenue desc, o_orderdate;
```

- HiveQL在语法上与SQL类似，还有很多特定的语法。在数据类型的支持上，除SQL中常见的数据结构外还支持数组、结构体、映射数据类型。在查询功能支持上，HiveQL支持嵌入MapReduce程序，用于处理复杂的任务。

【例4-67】HiveQL调用MapReduce程序。

```
FROM (
FROM docs
MAP doctext
USING 'python wordcount_mapper.py' as (word, cnt)
CLUSTER BY word ) it
REDUCE it.word, it.cnt USING 'python wordcount_reduce.py';
```

- Doctext是输入，word、cnt是Map程序的输出，cluster by对word哈希分区后作为Reduce程序的输入。

二、JSON数据管理

- JSON（JavaScript Object Notation，JS 对象标记）是一种轻量级的数据交换格式，它采用完全独立于编程语言的文本格式来存储和表示数据，已成为 Web 的通用语言，可供计算机跨众多软件和硬件平台进行快速分析和传输。在SQL:2016标准中增加了对JSON数据结构的支持，Oracle 12c、MySQL 5.7、SQL Server 2016等数据库增加了对JSON的数据管理功能，通过内置接口支持对JSON的存储、解析、查询、索引等功能，下面以SQL Server 2017为例演示对JSON数据的管理功能。

1.解析JSON数据

【例4-68】通过SQL命令解析JSON数据。

```
DECLARE @jsonVariable NVARCHAR(MAX)
SET @jsonVariable = N'[
        {
                "id":0,
          "Location": {
            "Horizontal_region":"Eastern hemisphere",
            "Vertical_region":"Sourthern Hemisphere"
          },
          "Population_B":0.78,
          "Area_million_km2": 30.37
        },
```

```
{
                "id":1,
  "Location": {
    "Horizontal_region":"Western hemisphere",
    "Vertical_region":"Northern Hemisphere"
  },
  "Population_B":0.822,
  "Area_million_km2": 42.07
},
            {
              "id":2,
  "Location": {
    "Horizontal_region":"Eastern hemisphere",
    "Vertical_region":"Northern Hemisphere"
  },
  "Population_B":3.8,
  "Area_million_km2": 44
},
```

```
    {
      "id":3,
    "Location": {
      "Horizontal_region":"Western hemisphere",
      "Vertical_region":"Northern Hemisphere"
    },
    "Population_B":0.8,
    "Area_million_km2":10.16
  },
    {
      "id":4,
    "Location": {
      "Horizontal_region":"Eastern hemisphere",
      "Vertical_region":"Northern Hemisphere"
    },
    "Population_B":0.36,
    "Area_million_km2": 6.5
  }
]'
```

第6节 面向大数据管理的SQL扩展语法

```
SELECT *
FROM OPENJSON(@jsonVariable)
  WITH (id int 'strict $.id',
        Location_Horizontal nvarchar(50) '$.Location.Horizontal_region',
Location_Vertical nvarchar(50) '$.Location.Vertical_region',
        Population_B real, Area_million_km2 real);
```

– SQL解析：通过内置OPENJSON函数解析JSON数据，使用WITH子句设置JSON数据解析结构。

2.JSON数据转换为关系数据

【例4-69】通过SQL命令将JSON数据插入表中。

- 将上例中SQL命令改写为：

```
SELECT * into region_json
FROM OPENJSON(@jsonVariable)
  WITH (id int 'strict $.id',
        Location_Horizontal nvarchar(50) '$.Location.Horizontal_region',
Location_Vertical nvarchar(50) '$.Location.Vertical_region',
        Population_B real, Area_million_km2 real);
```

- SQL解析：将解析出的JSON数据插入表region_json中。OPENJSON将JSON值转换为WITH短语定义的数据类型。

3.JSON数据更新为关系数据列

【例4-70】通过SQL命令将JSON数据插入表中的列。

- （1）在REGION表中增加一个JSON数据列。

```
alter table R1 add json_col NVARCHAR(MAX);
```

- （2）将JSON数据更新到JSON列中。

```
DECLARE @jsonVariable0 NVARCHAR(MAX)
SET @jsonVariable0 = '{"id":0,"Location":{"Horizontal_region":"Eastern hemisphere",
"Vertical_region":"Sourthern Hemisphere"},
"Population_B":0.78,"Area_million_km2":30.37}'

DECLARE @jsonVariable1 NVARCHAR(MAX)
SET @jsonVariable1 ='{"id":1,"Location":{"Horizontal_region":"Western hemisphere",
 "Vertical_region":"Northern Hemisphere"},
"Population_B":0.822,"Area_million_km2":42.07}'
```

```
DECLARE @jsonVariable2 NVARCHAR(MAX)
SET @jsonVariable2 = '{"id":2,"Location":{"Horizontal_region":"Eastern hemisphere",
"Vertical_region":"Northern Hemisphere"},
"Population_B":3.8,"Area_million_km2":44}’

DECLARE @jsonVariable3 NVARCHAR(MAX)
SET @jsonVariable3 = '{"id":3,"Location":{"Horizontal_region":"Western hemisphere",
"Vertical_region":"Northern Hemisphere"},
"Population_B":0.8,"Area_million_km2":10.16}’

DECLARE @jsonVariable4 NVARCHAR(MAX)
SET @jsonVariable4 = '{"id":4,"Location":{"Horizontal_region":"Eastern hemisphere",
"Vertical_region":"Northern Hemisphere"},
"Population_B":0.36,"Area_million_km2":6.5}'
```

```
update REGION set json_col=@jsonVariable0 where r_regionkey=0;
update REGION set json_col=@jsonVariable1 where r_regionkey=1;
update REGION set json_col=@jsonVariable2 where r_regionkey=2;
update REGION set json_col=@jsonVariable3 where r_regionkey=3;
update REGION set json_col=@jsonVariable4 where r_regionkey=4;
```

- （3）查看表中关系与JSON数据。

```
SELECT
r_name,
JSON_VALUE(json_col, '$.Location.Horizontal_region') AS Loca_H,
JSON_VALUE(json_col, '$.Location.Vertical_region') AS Loca_V,
JSON_VALUE(json_col, '$.Population_B') AS People,
JSON_VALUE(json_col, '$.Area_million_km2') AS Area
FROM REGION;
```

– SQL解析：JSON_VALUE函数用于从JSON字符串中解析值。

	R_REGIONKEY	R_NAME	R_COMMENT	json_col
1	0	AFRICA	special T...	{"id":0,"Location":{"Hori...
2	1	AMERICA	even, iro...	{"id":1,"Location":{"Hori...
3	2	ASIA	silent, b...	{"id":2,"Location":{"Hori...
4	3	EUROPE	special, ...	{"id":3,"Location":{"Hori...
5	4	MIDDLE EAST	furiously...	{"id":4,"Location":{"Hori...

	r_name	Loca_H	Loca_V	People	Area
1	AFRICA	Eastern hemisphere	Sourthern Hemisphere	0.78	30.37
2	AMERICA	Western hemisphere	Northern Hemisphere	0.822	42.07
3	ASIA	Eastern hemisphere	Northern Hemisphere	3.8	44
4	EUROPE	Western hemisphere	Northern Hemisphere	0.8	10.16
5	MIDDLE EAST	Eastern hemisphere	Northern Hemisphere	0.36	6.5

图4-32 JSON列和JSON解析

4.在SQL查询中使用关系和JSON数据

【例4-71】使用JSON数据执行SQL查询。

```
SELECT R.R_NAME, Detail.Loca_H, Detail.Loca_V, Detail.People,Detail.Area
FROM   REGION AS R
        CROSS APPLY
    OPENJSON (R.json_col)
        WITH (
          Loca_H   varchar(50) N'$.Location.Horizontal_region',
          Loca_V   varchar(50) N'$.Location.Vertical_region',
          People   real        N'$.Population_B',
          Area     real        N'$.Area_million_km2'
        )
 AS Detail
WHERE ISJSON(json_col)>0 AND Detail.People>0.8
ORDER BY JSON_VALUE(json_col,'$.Area_million_km2');
```

- SQL解析：OPENJSON函数用于将JSON数据转换为关系数据格式，使用JSON表达式用于不同的查询子句。

5. JSON索引

【例4-72】使用JSON数据执行SQL查询。

- 当查询中使用JSON值作为过滤条件查询时，可以为JSON值创建索引。

```
SELECT R_NAME,
JSON_VALUE(json_col, '$.Location.Horizontal_region') AS Loca_H,
JSON_VALUE(json_col, '$.Location.Vertical_region') AS Loca_V,
JSON_VALUE(json_col, '$.Population_B') AS People,
JSON_VALUE(json_col, '$.Area_million_km2') AS Area
FROM REGION
WHERE JSON_VALUE(json_col,'$.Location.Horizontal_region')='Eastern
hemisphere';
```

- 为JSON属性值创建索引需要如下步骤：

1.表中创建一个虚拟列，返回JSON中检索的属性值；

2.在虚拟列上创建索引。

```
ALTER TABLE R1
ADD vHorizontal_region AS JSON_VALUE(json_col,
'$.Location.Horizontal_region');
CREATE INDEX idx_json_Horizontal_region ON R1(vHorizontal_region);
```

6.关系数据库输出为JSON数据格式

【例4-73】将NATION表输出为JSON数据格式。

```
select * from NATION FOR JSON AUTO;
select * from NATION FOR JSON PATH,ROOT('Nations');
```

- SQL解析：FOR JSON AUTO将select语句结果自动输出为JSON数据格式；FOR JSON PATH可以增加嵌套结构和创建包装对象，如ROOT('Nations')在JSON数据中增加名字为Nations的根节点。

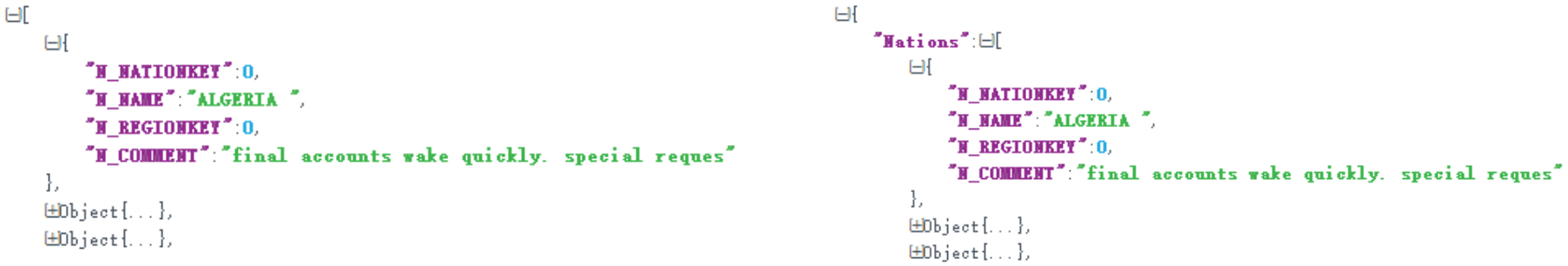


图4-33 FOR JSON AUTO与FOR JSON PATH输出结构

三、图数据管理

- 图数据管理技术起源于20世纪70年代，后来逐渐被关系数据库所取代。随着二十一世纪语义网技术的发展以及社交网络等大图数据应用的快速增长，图数据管理技术重新成为热点。图数据库是NoSQL数据库的一种类型，与关系数据库不同，图数据库采用图理论存储实体之间的关系信息，如社会网络中人与人之间的关系。
- 图数据库由一系列节点（或顶点）和边（或关系）组成。如图4-34所示，节点Person、City、Restaurant表示的实体，边LivesIn、FriendOf、Likes、LocatedIn表示连接的两个节点之间的关系，如朋友关系、居住于、喜欢等。

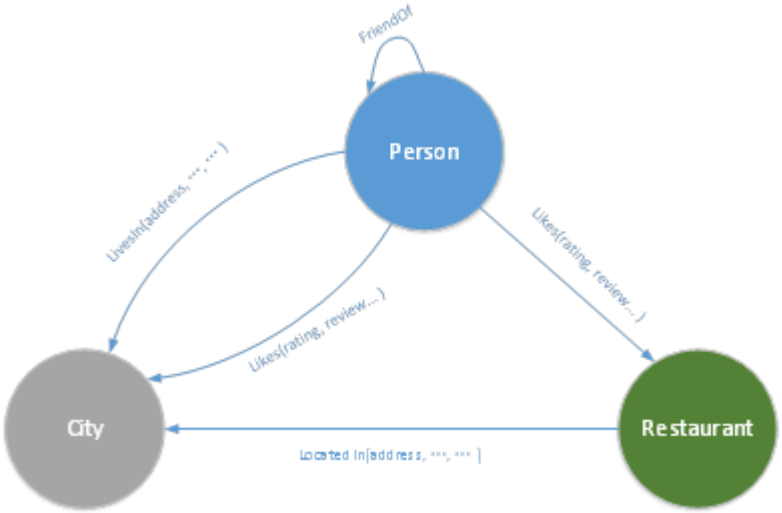


图4-34 图数据库示例

第6节 面向大数据管理的SQL扩展语法

- 节点和边都可能具有相关的属性。图形数据库的特征如下：
  - 由节点和边构成，节点和边可以有属性
  - 边有名字和方向，单条边可以灵活地连接图数据库中的多个节点
  - 图数据库易于表达模式匹配或多跳导航查询
- SQL Server 2017增加了对图数据库的支持。图数据库是一种抽象的数据类型，通过一组顶点节点、点和边来表现关系和连接，可以使用简单的方式来查询和遍历实体间的关系。下面以SQL Server 2017为例演示对图数据的管理功能。

1.创建图数据库

【例4-74】创建示例图数据库graphdemo。

```
CREATE DATABASE graphdemo;
USE  graphdemo;
```

- SQL命令解析：创建图数据库graphdemo并打开graphdemo。

```
CREATE TABLE Person (
  ID INTEGER PRIMARY KEY,
  name VARCHAR(100)
) AS NODE;
CREATE TABLE Restaurant (
  ID INTEGER NOT NULL,
  name VARCHAR(100),
  city VARCHAR(100)
) AS NODE;
CREATE TABLE City (
  ID INTEGER PRIMARY KEY,
  name VARCHAR(100),
  stateName VARCHAR(100)
) AS NODE;
```

- SQL解析：创建节点表Person、Restaurant、City。

```
CREATE TABLE likes (rating INTEGER) AS EDGE;
CREATE TABLE friendOf AS EDGE;
CREATE TABLE livesIn AS EDGE;
CREATE TABLE locatedIn AS EDGE;
```

- SQL解析：创建边表likes、friendOf、livesIn、locatedIn，rating为边likes的属性。

2.插入图数据[18]

【例4-75】通过insert命令在图数据库graphdemo中构造示例数据。

```
INSERT INTO Person VALUES (1,'John');
INSERT INTO Person VALUES (2,'Mary');
INSERT INTO Person VALUES (3,'Alice');
INSERT INTO Person VALUES (4,'Jacob');
INSERT INTO Person VALUES (5,'Julie');
```

- SQL解析：在节点表Person中插入示例数据。
- [18] https://docs.microsoft.com/zh-cn/sql/relational-databases/graphs/sql-graph-sample?view=sql-server-2017

```
INSERT INTO Restaurant VALUES (1,'Taco Dell','Bellevue');
INSERT INTO Restaurant VALUES (2,'Ginger and Spice','Seattle');
INSERT INTO Restaurant VALUES (3,'Noodle Land', 'Redmond');
```

- SQL解析：在节点表Restaurant中插入示例数据。

```
INSERT INTO City VALUES (1,'Bellevue','wa');
INSERT INTO City VALUES (2,'Seattle','wa');
INSERT INTO City VALUES (3,'Redmond','wa');
```

- SQL解析：在节点表City中插入示例数据。

```
INSERT INTO likes VALUES ((SELECT $node_id FROM Person WHERE id = 1),
    (SELECT $node_id FROM Restaurant WHERE id = 1),9);
INSERT INTO likes VALUES ((SELECT $node_id FROM Person WHERE id = 2),
    (SELECT $node_id FROM Restaurant WHERE id = 2),9);
INSERT INTO likes VALUES ((SELECT $node_id FROM Person WHERE id = 3),
    (SELECT $node_id FROM Restaurant WHERE id = 3),9);
INSERT INTO likes VALUES ((SELECT $node_id FROM Person WHERE id = 4),
    (SELECT $node_id FROM Restaurant WHERE id = 3),9);
INSERT INTO likes VALUES ((SELECT $node_id FROM Person WHERE id = 5),
    (SELECT $node_id FROM Restaurant WHERE id = 3),9);
```

- SQL解析：在边表likes中插入示例数据，需要为边likes的列$from_id和$to_id设置$node_id值。

```
INSERT INTO livesIn VALUES ((SELECT $node_id FROM Person WHERE id = 1),
    (SELECT $node_id FROM City WHERE id = 1));
INSERT INTO livesIn VALUES ((SELECT $node_id FROM Person WHERE id = 2),
    (SELECT $node_id FROM City WHERE id = 2));
INSERT INTO livesIn VALUES ((SELECT $node_id FROM Person WHERE id = 3),
    (SELECT $node_id FROM City WHERE id = 3));
INSERT INTO livesIn VALUES ((SELECT $node_id FROM Person WHERE id = 4),
    (SELECT $node_id FROM City WHERE id = 3));
INSERT INTO livesIn VALUES ((SELECT $node_id FROM Person WHERE id = 5),
    (SELECT $node_id FROM City WHERE id = 1));
```

- SQL解析：在边表livesIn中插入示例数据。

```
INSERT INTO locatedIn VALUES ((SELECT $node_id FROM Restaurant WHERE id = 1),
    (SELECT $node_id FROM City WHERE id =1));
INSERT INTO locatedIn VALUES ((SELECT $node_id FROM Restaurant WHERE id = 2),
    (SELECT $node_id FROM City WHERE id =2));
INSERT INTO locatedIn VALUES ((SELECT $node_id FROM Restaurant WHERE id = 3),
    (SELECT $node_id FROM City WHERE id =3));
```

- SQL解析：在边表locatedIn中插入示例数据。

```
INSERT INTO friendof VALUES ((SELECT $NODE_ID FROM person WHERE
ID = 1), (SELECT $NODE_ID FROM person WHERE ID = 2));
INSERT INTO friendof VALUES ((SELECT $NODE_ID FROM person WHERE
ID = 2), (SELECT $NODE_ID FROM person WHERE ID = 3));
INSERT INTO friendof VALUES ((SELECT $NODE_ID FROM person WHERE
ID = 3), (SELECT $NODE_ID FROM person WHERE ID = 1));
INSERT INTO friendof VALUES ((SELECT $NODE_ID FROM person WHERE
ID = 4), (SELECT $NODE_ID FROM person WHERE ID = 2));
INSERT INTO friendof VALUES ((SELECT $NODE_ID FROM person WHERE
ID = 5), (SELECT $NODE_ID FROM person WHERE ID = 4));
```

- SQL解析：在边表friendof中插入示例数据。

第6节 面向大数据管理的SQL扩展语法

- 查询节点表Person，其中$node_id是一个JSON字段，包含了实体类型和一个自增整型ID。查询边表locatedIn，其中有3个系统自动生成的列：$edge_id、$from_id、$to_id，$edge_id是包含实体类型和自增整型ID的JSON字段，$from_id和$to_id是来自于节点表Restaurant和City的$node_id字段内容。

	$node_id_63A63A44F51D488AB356A831808AFEC3	ID	name
1	{"type":"node","schema":"dbo","table":"Person","id":0}	1	John
2	{"type":"node","schema":"dbo","table":"Person","id":1}	2	Mary
3	{"type":"node","schema":"dbo","table":"Person","id":2}	3	Alice
4	{"type":"node","schema":"dbo","table":"Person","id":3}	4	Jacob
5	{"type":"node","schema":"dbo","table":"Person","id":4}	5	Julie

图4-35 节点表Person

	$edge_id_C63A8DC958F245E7BADA97199B99B3F1	$from_id_4C9EF50528AA40BCB575A3AACC679EE6	$to_id_7495F5378CF24D62BFD24DC4533BBDD0
1	{"type":"edge","schema":"dbo","table":"locatedIn","id":0}	{"type":"node","schema":"dbo","table":"Restaurant","id":0}	{"type":"node","schema":"dbo","table":"City","id":0}
2	{"type":"edge","schema":"dbo","table":"locatedIn","id":1}	{"type":"node","schema":"dbo","table":"Restaurant","id":1}	{"type":"node","schema":"dbo","table":"City","id":1}
3	{"type":"edge","schema":"dbo","table":"locatedIn","id":2}	{"type":"node","schema":"dbo","table":"Restaurant","id":2}	{"type":"node","schema":"dbo","table":"City","id":2}

图4-36 边表locatedIn

3.图数据查询

- MATCH语句用于定义图数据搜索条件，MATCH语句只能在 SELECT 语句中作为 WHERE子句的一部分与图形点和边表一起使用。其语法结构为：MATCH (<graph_search_pattern>)，graph_search_pattern指定图数据中的搜索模式或遍历路径，使用 ASCII 图表语法来遍历图形中的路径。模式将按照所提供的箭头方向通过边从一个节点转到另一个节点。括号内提供边名或别名，节点名称或别名显示在箭头两端。箭头可以指向两个方向中的任意一个方向。
- 下面通过graphdemo数据库示例通过节点与边表查找好友的操作

【例4-76】查找好友。

```
SELECT Person2.name AS FriendName
FROM Person Person1, friendOf, Person Person2
WHERE MATCH(Person1-(friendOf)->Person2)
AND Person1.name = 'John';
```

- SQL解析：在节点表Person中查找与John有朋友关系边（friendOf）的节点。

【例4-77】查找好友的好友。

```
SELECT Person3.name AS FriendName
FROM Person Person1, friendOf, Person Person2, friendOf friend2, Person Person3
WHERE MATCH(Person1-(friendOf)->Person2-(friend2)->Person3)
AND Person1.name = 'John';
```

- SQL解析：在节点表Person中查找与John朋友的朋友，通过边别名两次遍历节点。

【例4-78】查找共同的好友。

```
SELECT Person1.name AS Friend1, Person2.name AS Friend2
FROM Person Person1, friendOf friend1, Person Person2,
        friendOf friend2, Person Person0
WHERE MATCH(Person1-(friend1)->Person0<-(friend2)-Person2);
```

- SQL解析：查找与Person0共同具有朋友关系的Person。

【例4-79】查找John喜欢的餐馆。

```
SELECT Restaurant.name
FROM Person, likes, Restaurant
WHERE MATCH (Person-(likes)->Restaurant)
AND Person.name = 'John';
```

- SQL解析：查找John喜欢的餐馆。

【例4-80】查找John喜欢的餐馆。

```
SELECT Restaurant.name
FROM Person, likes, Restaurant
WHERE MATCH (Person-(likes)->Restaurant)
AND Person.name = 'John';
```

- SQL解析：通过likes边表查找与John喜欢的餐馆。

【例4-81】查找John的朋友喜欢的餐馆。

```
SELECT Restaurant.name
FROM Person person1, Person person2, likes, friendOf, Restaurant
WHERE MATCH(person1-(friendOf)->person2-(likes)->Restaurant)
AND person1.name='John';
```

- SQL解析：通过friendOf边表查找John的朋友，再查找John朋友喜欢的餐馆。

【例4-82】查找喜欢的餐馆与居住地位于相同城市的人。

```
SELECT Person.name
FROM Person, likes, Restaurant, livesIn, City, locatedIn
WHERE MATCH (Person-(likes)->Restaurant-(locatedIn)->City
AND Person-(livesIn)->City);
```

- SQL解析：通过likes边表查找餐馆，通过locatedIn边表查找所处城市，同时满足居住在相同的城市。

第6节 面向大数据管理的SQL扩展语法

- **SQL扩展语言小结**
- SQL语言简洁、功能强大、扩展性强，不仅是关系数据库的标准语言，也越来越多地被大数据管理平台所支持。面对大数据管理技术发展趋势，一方面SQL从关系数据库走向大数据平台，另一方面SQL也融合了大数据非结构化数据管理方法，扩展SQL的大数据管理能力。数据库不断吸收新兴的NoSQL数据库技术，通过对JSON数据类型的支持提供非结构化数据管理，SQL Server2017内置的图数据管理进一步扩展了数据库对非关系数据模型的支持能力。

Q & A