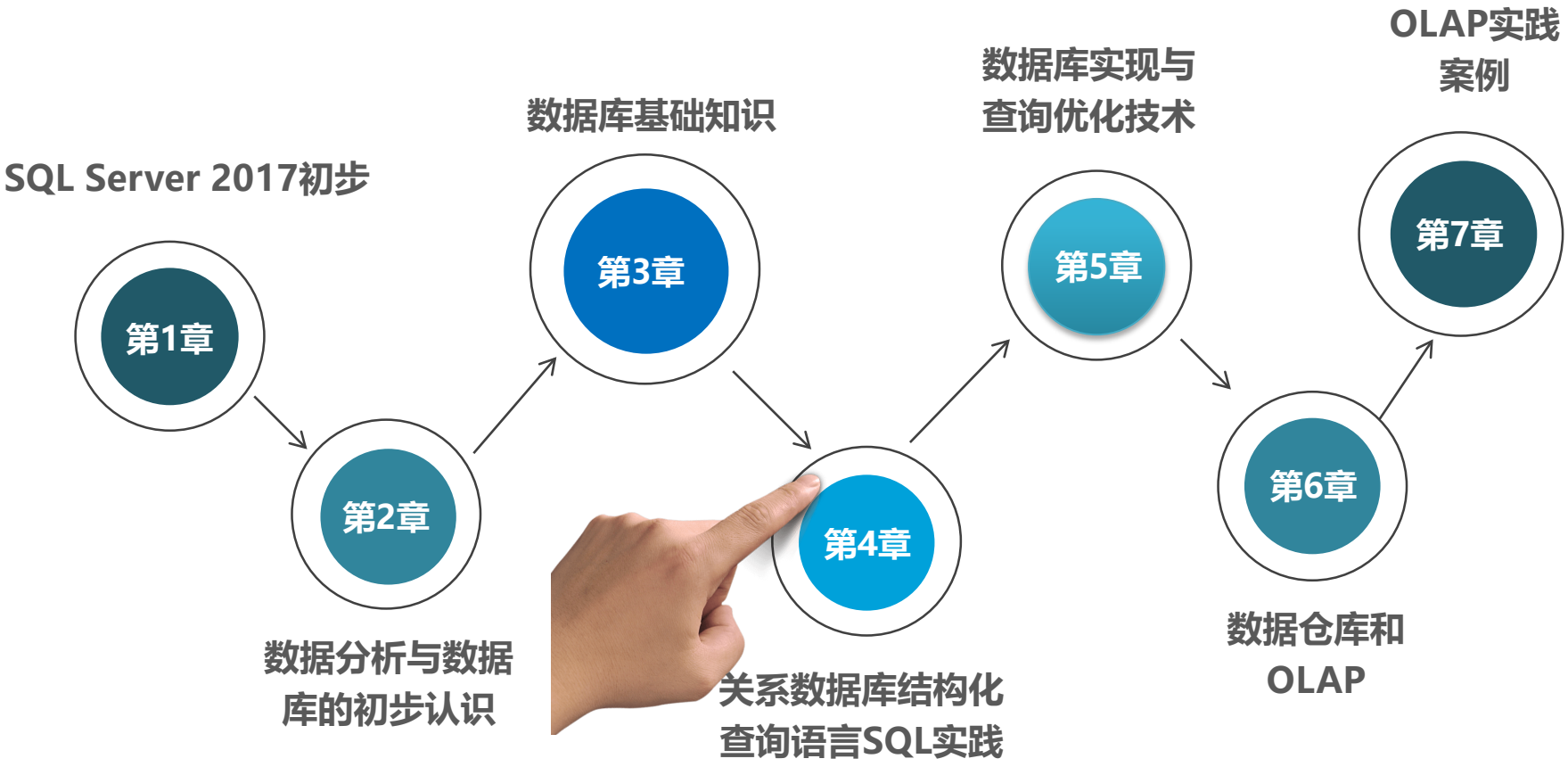
SQL Server 2017

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1节 SQL概述

- SQL语言是一种数据库查询和程序设计语言，允许用户在高层数据结构上工作，是关系数据库的标准语言，也是一个通用的、功能强大的关系数据库语言。SQL语言是高级的非过程化编程语言，不要求用户指定对数据的存放方法，也不需要用户了解具体的数据存放方式，这种特性保证了具有完全不同底层结构的数据库系统可以使用相同的SQL查询语言作为数据输入与管理的接口。SQL语言具有独立性，基本上独立于数据库本身、所使用的计算机系统、网络、操作系统等，基于SQL的DBMS产品可以运行在从个人机、工作站到基于局域网、小型机和大型机的各种计算机系统上，具有良好的可移植性。SQL语言具有共享性，数据库和各种产品都使用SQL作为共同的数据存取语言和标准的接口，使不同数据库系统之间的互操作有了共同的查询操作语言基础，能够实现异构系统、异构操作系统之间的共享与移植。SQL语言具有丰富的语义，其功能不仅仅是交互式数据操纵语言，还包括数据定义、数据库的插入/删除/修改等更新操作、数据库安全性完整性定义与控制、事务控制等功能，SQL语句可以嵌套，具有极大的灵活性并且能够表述复杂的语义。
- 本节所使用的示例数据库是TPC-H，SQL命令执行平台为SQL Server 2017。

第1节 SQL概述

一、SQL的产生与发展

- SQL 语言起源于1974年IBM公司圣约瑟研究实验室研制的大型关系数据库管理系统SYSTEM R中使用的SEQUEL语言（由BOYCE 和CHAMBERLIN 提出），后来在SEQUEL 的基础上发展了SQL 语言。
- 80年代初，美国国家标准局（ANSI）开始着手制定SQL标准，最早的ANSI标准于1986年完成，称为SQL86。标准的出台使SQL作为标准的关系数据库语言的地位得到加强。
- SQL标准几经修改和完善
  - 1992 年 制 定 了 SQL92 标 准 ， 全 名 是 “ International Standard ISO/IEC 9075:1992,Database Language SQL”。
  - SQL99则进一步扩展为框架、SQL基础部分、SQL调用接口、SQL永久存储模块、SQL宿主语言绑定、SQL外部数据的管理和SQL对象语言绑定等多个部分。
  - SQL2003 包 含 了 XML 相 关 内 容 ， 自 动 生 成 列 值 （ column values ） 。 SQL2006定义了结构化查询语言与XML（包含XQuery）的关联应用，
  - 2006年Sun公司将以结构化查询语言基础的数据库管理系统嵌入JavaV6。
  - SQL2008、SQL2011、SQL2016分别增加了一些新的语法、时序数据类型支持及对JSON等多样化数据类型的支持。

二、SQL语言结构

结构化查询语言包含6个部分：

（1）数据定义语言（DDL，Data Definition Language ）

DDL语句包括动词CREATE和DROP。在数据库中创建新表或删除表（CREAT TABLE 或DROP TABLE）；表创建或删除索引（CREATE INDEX或DROP INDEX）等。

（2）数据操作语言（DML，Data Manipulation Language）

DML语句包括动词INSERT、UPDATE和DELETE。分别用于插入、修改和删除表中的元组。

（3）事务处理语言（TPL，Transaction Control Language）

TPL语句能确保被DML语句影响的表的所有行能够得到可靠的更新。TPL语句包括BEGIN TRANSACTION、COMMIT和ROLLBACK。

（4）数据控制语言（DCL，Data Control Language）

DCL语句通过GRANT或REVOKE获得授权，分配或取消单个用户和用户组对数据库对象的访问权限。

（5）数据查询语言（DQL，Data Query Language）

DQL用于在表中查询数据。保留字SELECT是DQL（也是所有SQL）用得最多的动词，其他DQL常用的保留字有WHERE、ORDER BY、GROUP BY和HAVING等。

（6）指针控制语言（CCL，Cursor Control Language）

CCL语句用于对一个或多个表单独行的操作。例如DECLARE CURSOR、FETCH INTO和UPDATE WHERE CURRENT。

三、SQL语言特点

1．统一的数据操作语言：SQL语言集数据定义DDL、数据操纵DML和数据控制DCL语言于一体，可以完成数据库中的全部工作。

2．高度非过程化：与“面向过程”的语言不同，SQL进行数据操作时只提出要“做什么”，不必描述“怎么做”，也不需要了解存储路径。数据存储路径的选择及数据操作的过程由数据库系统的查询优化引擎自动完成，既减轻了用户的负担，又提高了数据的独立性。

3．面向集合的操作：SQL采用集合操作方式，即数据操作的对象是元组的集合，即关系操作的对象是关系，关系操作的输出也是关系。

4．使用方式灵活：SQL既可以以交互语言方式独立地使用，也可以作为嵌入式语言嵌入到C、C++、FORTRAN、JAVA、Python、R等主语言中使用。两种语言的使用方式相同，为用户提供了方便和灵活。

4．语法简洁，表达能力强，易于学习：在ANSI标准中，只包含了94个英文单词，核心功能只用9个动词，语法接近英语口语，易于学习。

- 数据查询：SELECT
- 数据定义：CREATE、DROP、ALTER
- 数据操纵：INSERT、DELETE、UPDATE
- 数据控制：GRANT、REVOKE

四、SQL数据类型

SQL语言中的五种主要的数据类型包括：字符型、文本型、数值型、逻辑型和日期型。

1.字符型

- 字符型用于字符串存储，根据字符串长度与存储长度的关系可以分为两大类：CHAR(n)和VARCHAR(n)，表示最大长度为n的字符串。
- CHAR(n)采用固定长度存储，当字符串长度小于宽度时尾部自动增加空格。VARCHAR(n)按照字符串实际长度存储，字符串需要加上表示字节长度值的前缀，当n不超过255时使用一个字节前缀数据，当n超过255时使用两个字节前缀数据。当字符串长度超过CHAR(n)或VARCHAR(n)的最大长度时，按n对字符串截断填充。

第1节 SQL概述

- 图4-1给出了字符串’ ’（空串，长度为0）、’ Hello’、’Hello World’、’Hello WorldCup’在CHAR(12)或VARCHAR(12)中的存储空间分配。
- CHAR(12)无论存储多长的字符串其长度都为12，VARCHAR(12)长度比实际字符串长度增加1个前缀数据字节，不同长度的字符串存储时实际使用的空间为n+1个字节。CHAR(n)会浪费一定的存储空间，但对于数据长度变化范围轻小的数据来说存储和访问简单，在列存储数据库中定长列易于实现根据逻辑位置访问数据物理地址；VARCHAR(n)在字符串长度变化范围较大时存储效率较高，但在存储管理和访问上较为复杂。

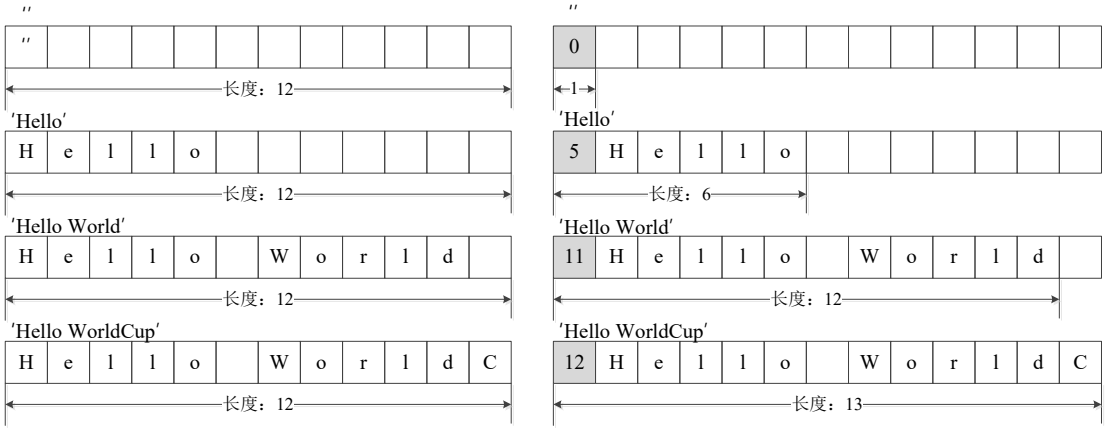


图4-1 CHAR(12)和VARCHAR(12)存储长度

- NCHAR(n)和NVARCHAR(n)数据类型采用Unicode标准字符集，Unicode标准用两个字节为一个存储单位。

3.数值型

- 数值型主要包括：整数、小数、浮点数、货币型。
- 整型包括：BIGINT、INT、SMALLINT和TINYINT。为了节省数据库的存储空间，在表设计时需要为列设置适合的数据类型，以免存储空间浪费。值域和存储空间如表2-2所示。

表4-1 BIGINT、INT、SMALLINT和TINYINT值域及存储空间

数据类型	范围	存储
BIGINT	-2^{63} (-9,223,372,036,854,775,808) 到 $2^{63}-1$ (9,223,372,036,854,775,807)	8 字节
INT	-2^{31} (-2,147,483,648) 到 $2^{31}-1$ (2,147,483,647)	4 字节
SMALLINT	-2^{15} (-32,768) 到 $2^{15}-1$ (32,767)	2 字节
TINYINT	0 到 255	1 字节

第1节 SQL概述

- NUMERIC型数据用于表示一个数的整数部分和小数部分。NUMERIC [(p[, s])]中，p表示不包含符号、小数点的总位数，范围是1~38，默认18；s表示小数位数，默认为0，满足0≤s≤p。NUMERIC型数据使用最大精度时可以存储从-1038+1到1038-1范围内的数。Decimal 与NUMERIC用法相同。
- 浮点型数据类型包括：REAL、FLOAT [(n)]、DOUBLE。REAL、FLOAT与DOUBLE分别对应单精度（4字节）与以精度浮点数（8字节），REAL 的 SQL-92 同义词为 FLOAT(24)，数值范围-3.40E + 38 至 -1.18E - 38、0 以及 1.18E - 38 至 3.40E + 38。FLOAT(n)类型n为用于存储 FLOAT 数值尾数，存储大小为4字节时n取值范围为1-24，精度是7位；DOUBLE PRECISION 的同义词为 FLOAT(53)，存储大小为8字节，n取值范围为25-53，精度是15位。FLOAT数值范围（取决于n值大小）-1.79E + 308 至 -2.23E - 308、0 以及 2.23E - 308 至 1.79E + 308。
- 浮点型数据属于近似数字数据类型，存储值的最近似值，并不存储指定的精确值。当要求精确的数字状态时，如银行、财务系统等应用中，不适合使用这类类型而是使用 INTEGER、DECIMAL、MONEY 或 SMALLMONEY 等数据类型。

- 货币型数据包括：MONEY 和SMALLMONEY。MONEY 和SMALLMONEY数据类型精确到它们所代表的货币单位的万分之一，各自的值域及存储空间如表4-2所示。

表4-2 MONEY 和SMALLMONEY值域及存储空间

数据类型	范围	存储
MONEY	-922,337,203,685,477.5808 到 922,337,203,685,477.5807	8 字节
SMALLMONEY	-214,748.3648 到 214,748.3647	4 字节

4.逻辑型

- 逻辑型BOOLEAN类型只能有两个取值：真（True）或假（False），用于表示逻辑结果。

5.日期型

日期型数据包含多种数据类型，如date、time、timestamp等，以DATETIME 和SMALLDATETIME为例说明日期型数据的存储大小与取值范围。

- 一个 DATETIME型的字段可以存储的日期范围是从1753年1月1日第一毫秒到9999年12月31日最后一毫秒，存储长度为8字节。
- SMALLDATETIME与DATETIME型数据同样使用，只不过它能表示的日期和时间范围比DATETIME型数据小，而且不如DATETIME型数据精确。一个SMALLDATETIME型的字段能够存储从1900年1月1日到2079年6月6日的日期，它只能精确到秒，存储长度为4字节。

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2节 数据定义SQL

- 数据库中的关系必须由数据定义语言（DDL）指定给系统，SQL的DDL用于定义关系及关系的一系列信息，包括：
  - 关系模式
  - 属性的值域
  - 完整性约束
  - 索引
  - 安全与权限
  - 存储结构
- SQL的数据定义功能包括模式、表、视图和索引。SQL标准通常不提供修改模式、修改视图和修改索引定义的操作，用户可以通过先删除原对象再重新建立的方式修改这些对象。

表4-3数据定义SQL命令

操作对象	操作方式		
	创建	删除	修改
模式	CREATE SCHEMA	DROP SCHEMA	
表	CREATE TABLE	DROP TABLE	ALTER TABLE
索引	CREATE INDEX	DROP INDEX	
视图	CREATE VIEW	DROP VIEW	

- 现代关系数据库管理系统提供层次化的数据库对角命名机制，最顶层是数据库（也称为目录），数据库中可以创建多个模式，模式中包括多个表、视图、索引等数据库对象。

第2节 数据定义SQL

一、模式的定义与删除

1.模式的定义

- 命令：CREATE SCHEMA
- 功能：创建一个新模式。模式是形成单个命名空间的数据库实体的集合，模式中包含表、视图、索引、权限定义等对象。该命令需要获得数据库管理员权限，或者用户被授予CREATE SCHEMA权限。
- 语法：

  CREATE SCHEMA schema_name [AUTHORIZATION username] [schema_element [...]]

  CREATE SCHEMA AUTHORIZATION username [schema_element [...]]
- SQL命令描述：
  - 模式名schema_name省略时使用用户名作为模式名，用户名username缺省时使用执行命令的用户名，只有超级用户才能创建不属于自己的模式。模式成员schema_element定义了要在模式中创建的对象，包含CREATE TABLE、CREATE VIEW和 GRANT命令创建的对象，其他对象可以在创建模式后独立创建。
  - 模式是数据库的命名空间，模式内的对象命名唯一，但可以与其他模式内的对象重名。当创建模式的用户需要被删除时，可以通过转让模式的所有权实现用户与模式的分离，避免因删除用户而导致的数据丢失问题。

第2节 数据定义SQL

SQL命令示例：

【例4-1】创建一个模式TPCHDEMO，授权给用户TPCH_user，并且在模式里面创建表和视图。

```
CREATE SCHEMA TPCHDEMO AUTHORIZATION TPCH_user
CREATE TABLE part (p_partkey int, p_name varchar(22), p_category varchar(7))
CREATE VIEW part_view AS
    SELECT p_name, p_category FROM part WHERE p_partkey <200;
```

上面的SQL命令与以下三个SQL命令等价：

```
CREATE SCHEMA TPCHDEMO;
CREATE TABLE TPCHDEMO.part (p_partkey int, p_name varchar(22), p_category varchar(7));
CREATE VIEW TPCHDEMO.part_view AS
    SELECT p_name, p_category FROM part WHERE p_partkey <200;
```

- 首先创建模式TPCHDEMO，然后创建以TPCHDEMO为前缀的表part和视图part_view。也就是说用户在创建模式的同时可以在模式中进一步创建表、视图，定义授权等。
- 当没有指定模式名时，模式名隐含为用户名TPCH_user，如：

```
CREATE SCHEMA AUTHORIZATION TPCH_user;
```

2.删除模式

- 命令：DROP SCHEMA
- 功能：删除指定模式。
- 语法：

  DROP SCHEMA schema_name;
- SQL命令描述：
  - 当删除模式时，如果模式中已经定义了下属的数据库对象，则中止该删除模式语句的执行，需要首先将模式内的对象删除，然后才能将模式删除。

【例4-2】删除模式TPCHDEMO.

删除模式TPCHDEMO时需要首先删除表part和视图part_view，然再删除模式TPCHDEMO：

```
DROP TABLE TPCHDEMO.part;
DROP VIEW TPCHDEMO.part_view;
DROP SCHEMA TPCHDEMO;
```

3.模式转移

- 模式转移命令用于将一个模式中的数据库对象转换给另一个模式。
- 【例4-3】创建一个模式temp并在模式中创建表users，然后将users转移给模式TPCHDEMO。

```
CREATE SCHEMA temp
CREATE TABLE users (id INT, username VARCHAR(30));
ALTER SCHEMA TPCHDEMO TRANSFER OBJECT::temp.users;
```

图4-2模式转移

- 首先创建模式temp和模式中的表users，在SQL Server 2017管理器中查看数据库中的表对象存在名称为temp.users的表。通过ALTER SCHEMA命令将模式temp中的表users转移给模式dbo，命令执行完后查询管理器确认表名称改为dbo.users，实现了模式中对象的转移。

二、表的定义、删除与修改

1.定义表

- 命令：CREATE TABLE
- 功能：创建一个基本表。
- 语法：

CREATE TABLE [database_name . [schema_name] . | schema_name .] table_name
(<column_name> < type_name > [constraint_name]
[,<column_name> < type_name > [constraint_name]]
……
[,<table_constraint >]);

第2节 数据定义SQL

- **SQL命令描述：**
  - 基本表是关系的物理实现。表名table_name定义了关系的名称，相同的模式中表名不能重复，不同的模式或数据库之间表名可以相同。列名column_name是属性的标识，表中的列名不能相同，不同表的列名可以相同。当查询中所使用不同表的列名相同时，需要使用“表名.列名”来标识相同名称的列，当列名不同时，不同表的列可以直接通过列名访问，因此在标准化的设计中通常采用表名缩写通过下划线与列名组成复合列名的命名方式来唯一标识不同的列，如part表的name列命名为“p_name”，supplier表的name列命名为“s_name”，通过表名缩写前缀来区分不同表中的列。
  - 数据类型type_name规定了列的取值范围，需要根据列的数据特点定义适合的数据类型，既要避免因数据类型值域过小引起的数据溢出问题，也需要避免因数据类型值域过大导致的存储空间浪费问题。在大数据存储时，数据类型的宽度决定了数据存储空间，需要合理地根据应用的特征选择适当的数据类型。

第2节 数据定义SQL

- 列级完整性约束constraint_name包括：
  - 列是否可以取空值：
    - [NULL | NOT NULL]
    - 如P_SIZE int NULL表示列P_SIZE可以取空值。注意，表中设置为主码的列不可为空，需要设置NOT NULL约束条件。
  - 列是否为主键/唯一键：
    - { PRIMARY KEY | UNIQUE } [CLUSTERED | NONCLUSTERED]
    - 如S_SUPPKEY int PRIMARY KEY CLUSTERED表示列S_SUPPKEY设置为主码并创建聚集索引。聚集索引是指表中行数据的物理顺序与键值的逻辑（索引）顺序相同，一个表只能有一个聚集索引，一些数据库系统默认为主码建立聚集索引。
  - 列是否是外码：
    - REFERENCES [schema_name.] referenced_table_name [(ref_column)]
- 如N_REGIONKE int REFERENCES REGION (R_REGIONKEY)表示列N_REGIONKE是外码，参照表REGION中的列R_REGIONKEY。约束条件定义在列之后的方式称为列级约束。
- 表级约束table_constraint是为表中的列所定义的约束。当表中使用多个属性的复合主码时，主码的定义需要使用表级约束。列级参照完整性约束也可以表示为表级参照完整性约束。

第2节 数据定义SQL

【例4-4】参照图3-13（A）模式，写出TPC-H数据库中各表的定义命令。

```
CREATE TABLE REGION
(       R_REGIONKEY     integer         PRIMARY KEY,
        R_NAME          char(25),
        R_COMMENT       varchar(152) );

CREATE TABLE NATION
(       N_NATIONKEY     integer         PRIMARY KEY,
        N_NAME          char(25),
        N_REGIONKEY  integer     REFERENCES REGION (R_REGIONKEY),
        N_COMMENT       varchar(152)  );
```

第2节 数据定义SQL

```
CREATE TABLE PART
(       P_PARTKEY       integer         PRIMARY KEY,
        P_NAME          varchar(55),
        P_MFGR          char(25),
        P_BRAND         char(10),
        P_TYPE          varchar(25),
        P_SIZE          integer,
        P_CONTAINER     char(10),
        P_RETAILPRICE   decimal,
        P_COMMENT               varchar(23) );

CREATE TABLE SUPPLIER
(       S_SUPPKEY               integer         PRIMARY KEY,
        S_NAME          char(25),
        S_ADDRESS               varchar(40),
        S_NATIONKEY     integer         REFERENCES NATION
(N_NATIONKEY),
        S_PHONE         char(15),
        S_ACCTBAL               decimal,
        S_COMMENT               varchar(101) );
```

第2节 数据定义SQL

```
CREATE TABLE PARTSUPP
(       PS_PARTKEY              integer         REFERENCES PART
(P_PARTKEY),
        PS_SUPPKEY              integer         REFERENCES SUPPLIER
(S_SUPPKEY),
        PS_AVAILQTY             integer,
        PS_SUPPLYCOST    Decimal,
        PS_COMMENT       varchar(199),
        PRIMARY KEY(PS_PARTKEY, PS_SUPPKEY) );

CREATE TABLE CUSTOMER
(       C_CUSTKEY               integer         PRIMARY KEY,
        C_NAME           varchar(25),
        C_ADDRESS               varchar(40),
        C_NATIONKEY             integer         REFERENCES NATION
(N_NATIONKEY),
        C_PHONE          char(15),
        C_ACCTBAL               Decimal,
        C_MKTSEGMENT     char(10),
        C_COMMENT        varchar(117) );
```

第2节 数据定义SQL

```
CREATE TABLE ORDERS
(       O_ORDERKEY              integer         PRIMARY KEY,
        O_CUSTKEY               integer   REFERENCES CUSTOMER
(C_CUSTKEY),
        O_ORDERSTATUS   char(1),
        O_TOTALPRICE    Decimal,
        O_ORDERDATE     Date,
        O_ORDERPRIORITY         char(15),
        O_CLERK         char(15),
        O_SHIPPRIORITY  integer,
        O_COMMENT       varchar(79)  );
```

第2节 数据定义SQL

```
CREATE TABLE LINEITEM
(       L_ORDERKEY                  integer         REFERENCES ORDERS
(O_ORDERKEY),
        L_PARTKEY           integer,
        L_SUPPKEY           integer,
        L_LINENUMBER         integer,
        L_QUANTITY          decimal,
        L_EXTENDEDPRICE     decimal,
        L_DISCOUNT          decimal,
        L_TAX                       decimal,
        L_RETURNFLAG        char(1),
        L_LINESTATUS        char(1),
        L_SHIPDATE          date,
        L_COMMITDATE        date,
        L_RECEIPTDATE       date,
        L_SHIPINSTRUCT      char(25),
        L_SHIPMODE          char(10),
        L_COMMENT           varchar(44),
        PRIMARY KEY(L_ORDERKEY, L_LINENUMBER),
        FOREIGN KEY (L_PARTKEY, L_SUPPKEY) REFERENCES PARTSUPP
(PS_PARTKEY,        PS_SUPPKEY));
```

第2节 数据定义SQL

- 在表定义命令中，单属性上的主码及参照完整性约束定义可以使用列级约束定义，直接写在列定义语句中，复合属性主、外码定义则需要使用表级定义，通过独立的语句写在表定义命令中。在参照完整性约束定义中，使用列级约束定义完整性约束时只需要定义参照的表及列，而使用表级约束定义完整性约束时则需要定义外码及参照的表和列。

2.修改表

- 命令：ALTER TABLE
- 功能：修改基本表。
- 语法：

  ALTER TABLE < table_name >

  [ADD <column_name> < type_name > [constraint_name]]

  [ADD <table_constraint>]

  [DROP <column_name>]

  [DROP <constraint_name >]

  [ALTER COLUMN < column_name > [type_name] | [NULL | NOT NULL]];
- SQL命令描述：
  - 修改基本表，其中ADD子句用于增加新的列（列名、数据类型、列级约束）和新的表级约束条件；DROP子句用于删除指定的列或者指定的完整性约束条件；ALTER TABLE子句用于修改原有的列定义，包括列名、数据类型等。

第2节 数据定义SQL

- SQL命令示例：

【例4 5】完成下面的表修改操作。

```
ALTER TABLE LINEITEM ADD L_SURRKEY int;
--SQL命令解析：增加一个int类型的列L_SURRKEY;
ALTER TABLE LINEITEM ALTER COLUMN L_QUANTITY SMALLINT;
--SQL命令解析：将L_QUANTITY列的数据类型修改为SMALLINT：
ALTER TABLE ORDERS ALTER COLUMN O_ORDERPRIORITY varchar(15)
NOT NULL;
--SQL命令解析：将O_ORDERPRIORITY列的约束修改为NOT NULL约束：
ALTER TABLE LINEITEM ADD CONSTRAINT FK_S FOREIGN KEY
(L_SURRKEY) REFERENCES SUPPLIER(S_SUPPKEY);
--SQL命令解析：在LINEITEM表中增加一个外键约束。CONSTRAINT关键字
定义约束的名称FK_S，然后定义表级参照完整性约束条件。
ALTER TABLE LINEITEM DROP CONSTRAINT FK_S;
--SQL命令解析：在LINEITEM表中删除外键约束FK_S。
ALTER TABLE LINEITEM DROP COLUMN L_SURRKEY;
--SQL命令解析：删除表中的列L_SURRKEY：
```

- 命令执行约束：
  - 在列的修改操作中，如果将数据宽度由小变大时可以直接在原始数据上修改，如果数据宽度由大变小或者改变数据类型时，通常需要先清除掉数据库中该列的内容然后才能修改。在这种情况下，可以通过临时列完成列数据类型修改时的数据交换，如修改列数据类型时先增加一个与被修改的列类型一样的列作为临时列，然后将要修改列的数据复制到临时列并置空要修改的列，然后修改该列的数据类型，再从临时列将数据经过数据类型转换后复制回被修改的列，最后删除临时列。
  - 当数据库中存储了大量数据时，表结构的修改会产生较高的列数据更新代价，因此需要在基本表的设计阶段全面考虑列的数量、数据类型和数据宽度，尽量避免对列的修改。

3.删除表

- 命令：DROP TABLE
- 功能：删除一个基本表。
- 语法：

  DROP TABLE < table_name > [RESTRICT|CASCADE];

- SQL命令描述：
  - RESTRICT：缺省选项，表的删除有限制条件。删除的基本表不能被其他表的约束所引用，如FOREIGN KEY，不能有视图、触发器、存储过程及函数等依赖于该表的对象，如果存在，需要首先删除这些对象或者解除与该表的依赖后才能删除该表。
  - CASCADE：无限制条件，表删除时相关对象一起删除。

第2节 数据定义SQL

- SQL命令示例：

【例4-6】删除表part。

```
DROP TABLE part;
```

- 不同的数据库对DROP TABLE命令有不同的规定，有的数据库不支持RESTRICT | CASCADE选项，在删除表时需要手工删除与表相关的对象或解除删除表与其他表的依赖关系。
- 对于依赖于基本表的对象，如索引、视图、存储过程和函数、触发器等对象，不同的数据库在删除基本表时采取的策略有所不同，通常来说，删除基本表后索引会自动删除，视图、存储过程和函数在不删除时也会失效，触发器和约束引用在不同数据库中有不同的策略。

4.内存存储模型

- 随着硬件技术的发展，大内存与多核处理器成为新一代数据库主流的高性能计算平台，内存数据库通过内存存储模型实现数据存储在高性能内存，从而显著提高查询处理性能。
- 以SQL Server Hekaton内存引擎为例，数据库可以创建内存优化表。在SQL Server 2016中创建内存表需要以下几个步骤：
  - 创建内存优化数据文件组并为文件组增加容器
  - 创建内存优化表
  - 导入数据到内存优化表

【例4-7】为TPC-H（TPCH）数据库的LINEITEM表创建内存表。

（1）为数据库TPCH创建内存优化数据文件组并为文件组增加容器

```
ALTER DATABASE TPCH ADD FILEGROUP TPCH_mod CONTAINS
MEMORY_OPTIMIZED_DATA
ALTER DATABASE TPCH ADD FILE (NAME='TPCH_mod', FILENAME=
'C:\IM_DATA\TPCH_mod') TO FILEGROUP TPCH_mod;
```

为数据库TPCH增加文件组TPCH_mod，为文件组增加文件C:\IM_DATA\TPCH_mod作为数据容器。

（2）创建内存表

- 通过下面的SQL命令创建内存表LINEITEM_IM，其中子句index ix_orderkey nonclustered hash (lo_orderkey,lo_linenumber) with (bucket_count=8000000)用于创建主键哈希索引，测试集（SF=1）LINEITEM表中记录数量约为600万，因此指定哈希桶数量为8000000。哈希桶数量设置较大能够提高哈希查找性能，但过大的哈希桶数量也会产生存储空间的浪费。
- 命令子句WITH (MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_ONLY)用于设置内存表类型，MEMORY_OPTIMIZED=ON表示创建表为内存表。DURABILITY = SCHEMA_ONLY表示创建非持久化内存优化表，不记录这些表的日志且不在磁盘上保存它们的数据，即这些表上的事务不需要任何磁盘I/O，但如果服务器崩溃或进行故障转移，则无法恢复数据；DURABILITY=SCHEMA_AND_DATA表示内存优化表是完全持久性的，整个表的主存储是在内存中，即为从内存读取表中的行，和更新这些行数据到内存中，但内存优化表的数据同时还在磁盘上维护着一个仅用于持久性目的副本，在数据库恢复期间，内存优化的表中的数据可以再次从磁盘装载。

- 以下为创建非持久化内存优化表示例：

```
CREATE TABLE LINEITEM_IM (
    L_ORDERKEY          integer         not null,
    L_PARTKEY           integer         not null,
    L_SUPPKEY           integer         not null,
    L_LINENUMBER        integer         not null,
    L_QUANTITY          decimal         not null,
    L_EXTENDEDPRICE     decimal         not null,
    L_DISCOUNT          decimal         not null,
    L_TAX               decimal         not null,
    L_RETURNFLAG        char(1)         not null,
    L_LINESTATUS        char(1)         not null,
    L_SHIPDATE          date            not null,
    L_COMMITDATE        date            not null,
    L_RECEIPTDATE       date            not null,
    L_SHIPINSTRUCT      char(25)        not null,
    L_SHIPMODE          char(10)        not null,
    L_COMMENT           varchar(44)     not null,
    index ix_orderkey nonclustered hash (L_ORDERKEY,L_LINENUMBER )
with (bucket_count=8000000)
) WITH (MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_ONLY);
```

三、代表性的索引技术

- 当表的数据量比较大时，查询操作需要扫描大量的数据而产生较大的耗时。索引是数据库中重要的性能优化技术，通过创建索引，在表上创建一个或多个索引，提供多种存储路径，数据库能够自动地执行索引查找，提高数据库的查询性能。关系数据库中常用的索引包括B+树索引、哈希索引、位图索引、位图连接索引、存储索引和列存储索引等。

1.索引类型

（1）聚集索引

- 数据库通常会为关系中定义的主码自动创建聚集索引（clustered index），即记录按主码的顺序物理存储，保持数据在逻辑上和物理上都能按主码成顺序访问，这种聚簇存储机制能够有效地提高查询性能，但一个关系上只能创建一个聚集索引。

（2）B+树索引

- B+树索引是磁盘数据库的一般经典索引结构，它的基本思想是以page为单位组织表记录键值-地址（PageId）对的分层存储。在创建索引时，原始表中较长的记录按索引的结构抽取出键值-地址对排序后以page为单位存储在B+树索引叶节点层，各叶节点形成链表结构，记录了索引键值的排序序列。每个叶节点page中的最小值抽取出构建上级非叶索引节点，以page为单位依次存储每个叶节点最小值-地址对数据，即叶节点为记录建立索引，非叶节点为下一级索引节点建立索引。非叶节点依次向上构建，直到只产生唯一的非叶节点作为B+树索引的根节点。
- 假设表记录宽度为80字节，记录行数为10亿条，索引列数据类型为int（4字节），page大小为4KB，则记录需要存储在80*1,000,000,000/(4*1024)≈19,531,250个page中，即，在没有索引的情况下如果查找某个键值对应的记录需要顺序扫描19,531,250个page。
- 在建立B+树索引时，10亿条记录键值-地址对宽度为8字节（假设键值4字节，地址4字节），则每个4KB大小的page中可以存储512个索引项，B+树索引的叶节点需要38147个page，叶节点中的每一个page中的最小值和PageId构成第二级非叶节点的索引项，需要74个二级非叶节点索引page，然后74个索引page中的最小值-地址对继续构造1个第三级根节点page。

- 在执行索引查找时，首先访问B+树索引的根节点，根据键值大小访问下一级非叶索引节点，再依次访问叶节点，获得键值匹配记录的地址（PageId），最后访问数据page，在page页面内顺序扫描，读取相应的记录。查找过程如图4-3所示键值为61记录的查找过程。

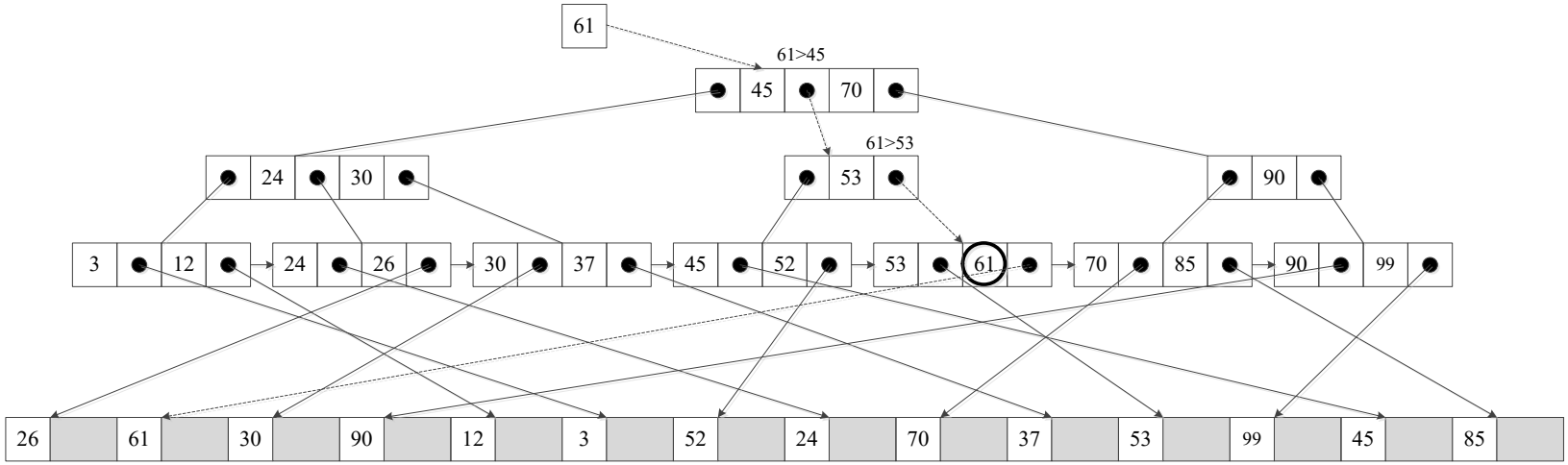


图4-3 B+树索引

- B+树索引在底层存储顺序的键值数据，比较适合进行点查找和范围查找，在范围查找时只需要查找范围表达式的最小值后在叶结点上顺序访问，直到叶结点对应的数据超过最大值为止。

（3）CSB+-Tree索引

- B+树索引是一种磁盘索引结构，在内存数据库中CSB+-Tree是一种cache敏感的内存B+-Tree索引（Cache-Sensitive B+-Tree）。内存索引优化技术的关键是提高索引查找过程的cache line利用率，即在一个cache line中存储尽可能多的索引信息。通常的B+-Tree节点中至少一有半的空间用于存储下级节点的指针，而CSB+-Tree将给定节点的下一级子节点在数组中连续存储，并且只保留第一个子节点的指针，其他子节点的地址通过第一个子节点地址加上偏移地址计算而得到。通过这种地址计算机制，CSB+-Tree的非叶节点中减少了指针的存储空间，能够存储更多的节点信息，从而提高cache line的利用率。
- 内存访问以cache line为单位，一个cache line长度通常为64字节，内存B+-Tree索引的节点以cache line为单位，假设一个键值和指针的长度均为4字节，则一个cache line节点中至多能存储7个子节点（7个键值，8个指针）信息，而CSB+-Tree节点中只存储一个下级子节点指针，则一个cache line节点中能够存储14个子节点（1个int型节点内键值数量字段，1个下级第一个子节点地址指针，14个键值）信息，从而提高了索引查找时cache line的利用率，降低了索引树的高度。

- 图4-4为CSB+-Tree示意图，虚线部分是连续存储的下级子节点组（node group），非叶节点上的箭头代表指向下一级子节点中第一个子节点的指针，节点组的所有节点物理相邻地存储在连续的地址区域中。图中CSB+-Tree节点组中包含不超过3个的节点，如节点[3]的下级节点有两个，节点[13 19]的下级节点有3个，下级节点定长连续存储，通过首节点地址和偏移地址可以计算出下一级每个节点的地址。

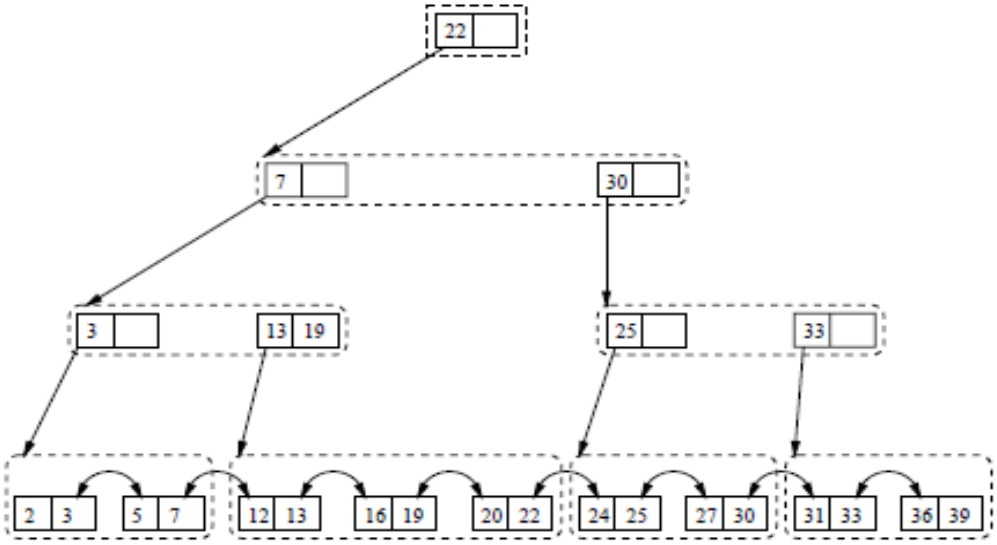
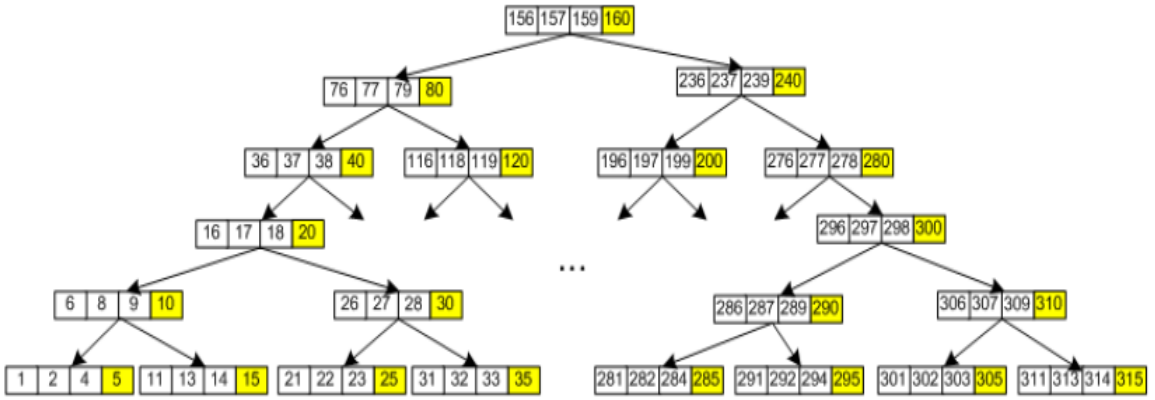


图4-4 CSB+-Tree索引

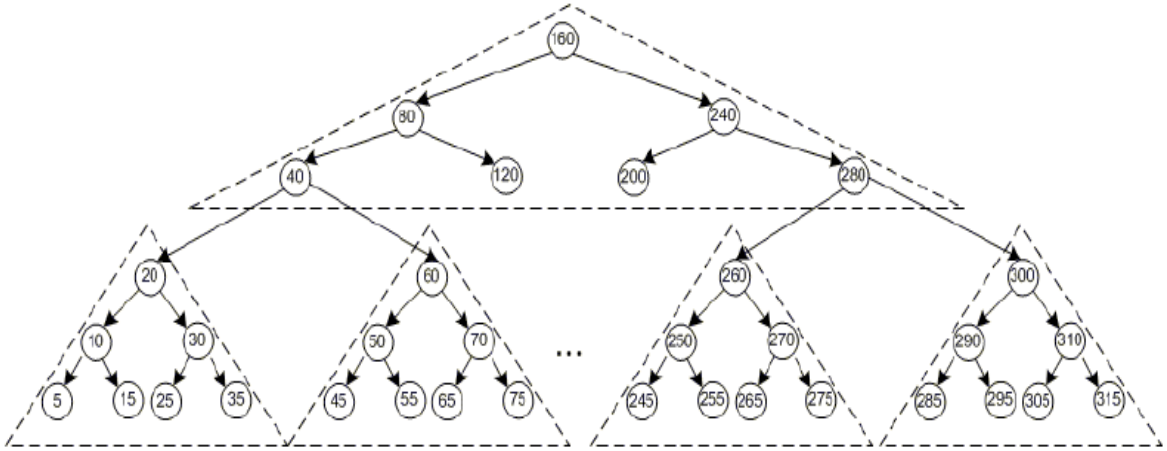
- B+-Tree节点分裂时只需要创建一个新的节点，而CSB+-Tree节点分裂时则需要创建一个新的节点组，当CSB+-Tree在更新时产生较多的分裂操作时，CSB+-Tree的维护代价较大，可以通过在CSB+-Tree节点中预留较大的空间来减少分裂代价。CSB+-Tree分裂时节点组的复制代价较大，进一步的优化策略是将节点组存储为多个段，插入数据产生分裂时通过段复制代替代价较大的节点组复制，从而降低CSB+-Tree索引维护代价。
- 总体上说，CSB+-Tree索引具有较好的查找性能，能够提高cache line利用率，提高索引性能。但由于采用偏移地址计算来代替节点地址存储的策略需要将节点的下一级节点组连续存储，因此产生较大的索引维护代价。CSB+-Tree索引适用于读密集型的决策支持负载，如SSB、TPC-H、TPC-DS等负载中索引字段为顺序递增的主键，使用CSB+-Tree索引能够获得较好的索引查找代价并且具有较低的索引维护代价。

（4）CST-Tree索引

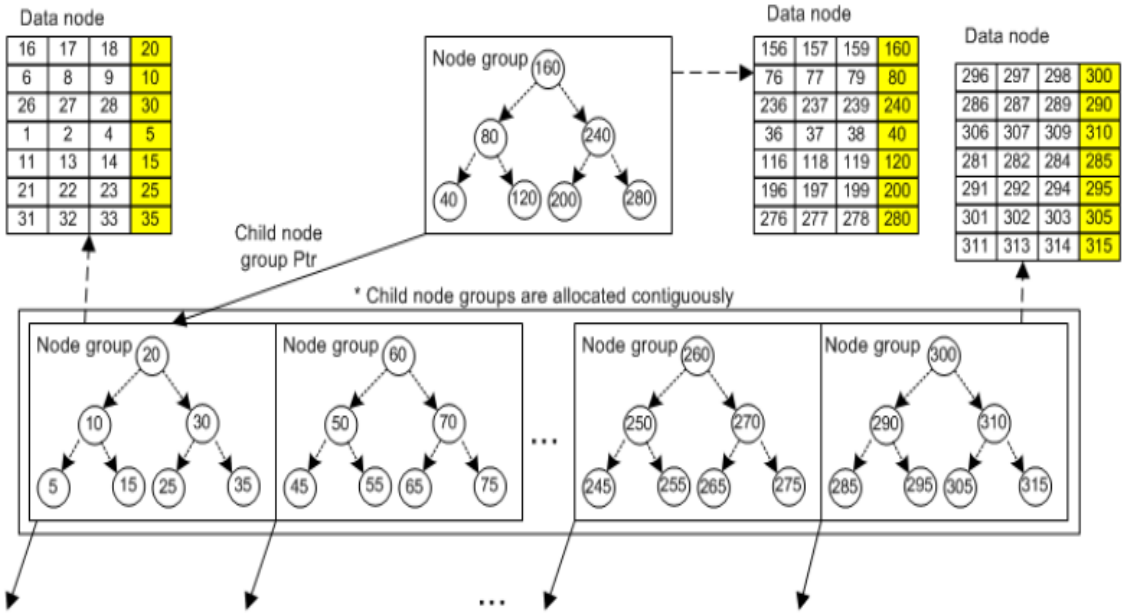
- T-Tree是AVL-Tree的变种，是一种适合内存存储的索引结构，它在一个节点中存储n个键值和左、右子树指针。相对于B+-Tree，T-Tree一个节点中只有两个指针，因此T-Tree的高度远远高于B+-Tree，从根节点到叶节点检索时的内存访问代价较高。提高T-Tree内存访问性能的另一个方法是按cache line大小设计T-Tree节点。在T-Tree检索中，每个节点的访问产生一个cache line miss，但在一个节点中通常只有最大值和最小值用于比较查找键值，cache line的利用率低，一个检索操作产生较多的cache line misses。
- 在图4-5（A）所示的T-Tree中，只有最大值用于节点查找。图4-5（B）为T-Tree节点的最大值构建一个二分查找树，该二分查找树用作T-Tree节点的索引结构，用于定位包含查找键值的T-Tree节点。该二分查找树相对于T-Tree只占用较少的存储空间，并且能够显著提高索引查找时的cache命中率。



（A）T-Tree索引



（B）T-Tree索引上的二分查找



（C）CST-Tree索引

44

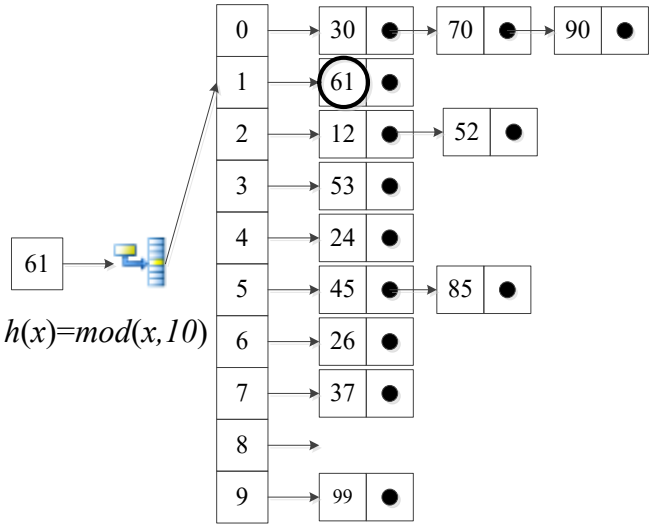
- 当二叉树存储为数组时，不需要存储父节点和子节点的指针，可以根据节点的位置i分别计算出其父节点、左子节点、右子节点的位置为i/2、i*2和i*2+1。如图4-5（B）中第一个二分查找子树中第3个节点240对应的父节点位置为1，左子节点位置为6，右子节点位置为7。当每个二分查找子树连续存储时，节点只需要存储下级第一个子树的地址指针，其他子树可以通过偏移地址计算得到其存储地址。图4-5（C）使用cache line长度的数组存储节点组，当键值为4字节，cache line长度为32字节时，一个节点组的二分查找树包含7个键值，高度为3。节点组内部的查找只产生一个cache line miss，节点组之间的查找产生新的cache line miss。
- 例如在T-Tree索引中查找287时，首先在根节点组中查找，通过一次cache line访问在节点组内部完成与3个节点160、240和280的比较，然后访问另一个节点组，产生一个cache line miss；在图4-5（C）最右侧的节点组中与节点300和290进行比较，确定键值287在节点290对应的T-Tree节点中，访问对应的T-Tree节点并找到匹配的键值。

- 当CST-Tree插入新的数据产生节点分裂时，需要增加新的节点以及新的节点组。
- CST-Tree和CSB+-Tree索引一方面以cache line为节点存储单位，减少索引查找时的cache line miss数量，另一方面采用连续的数组存储消除指针存储代价，提高cache line的利用率，提高索引的cache性能。但CST-Tree和CSB+-Tree索引需要保证节点组的连续存储，当数据更新产生较多的节点分裂时，索引维护代价较传统索引更高，因此适用于更新较少或者按索引键值顺序更新的应用场景。

（5）哈希索引

- B+树索引是一种分层的、键值有序的索引结构，可以支持点查询和范围查询。哈希索引主要支持点查询，通过哈希函数建立键值与地址之间的直接映射关系，如图4-6所示的哈希索引结构。哈希索引对应随机查找过程，索引的效率受随机访问性能的影响较大，如通过哈希索引访问较多的键值时，每个索引访问都可能导致一个磁盘page的随机访问，产生大量随机磁盘I/O，索引访问性能在选择率较高时可能低于不使用索引的顺序扫描方法。提高哈希索引性能的方法是将哈希表建立在高性能随访问存储设备中，如通过数据分区技术将表划分为较小的分区，在较小的分区上建立哈希表，使哈希表位于内存或容量更小、性能更高的cache中，从而提高哈希索引访问性能。

- 哈希查询适合于进行等值查找，通过哈希函数计算直接得到数据存储位置，在等值查找效率上比B+树索引要高，但不支持范围查找。



（6）位图索引

- 位图索引是一种通过位图记录属性列在每个成员在行中位置的技术，位图索引适用于属性列中不同成员的数量与行数之比小于1%的低势集属性。如图4-7所示，关系中的属性Gender和Country分别有2个和3个成员，为属性Gender创建的位图索引包含两个位图，分别表示Gender值为M或F的记录在表中的位置。图右所示的位图索引结构可以看作是为属性Gender按属性成员的数量创建一个位图矩阵，Gender属性的每一行的取值在Gender位图矩阵的每一行中只有一个对应位置取值为1，其余位置取值为0。通过位图索引机制，Gender属性存储为2个位图，压缩了属性存储空间，而且通过位图索引能够直接获得Gender取指定值时所有满足条件记录的位置。

Customer_id	Gender	Country
1	F	Mexico
2	M	Canada
3	F	USA
4	M	Canada
5	F	USA
6	F	USA
7	F	Mexico
8	M	Mexico
9	M	Canada
10	M	USA
11	M	USA

	Gender		Country		
customer_id	M	F	Mexico	Canada	USA
1	0	1	1	0	0
2	1	0	0	1	0
3	0	1	0	0	1
4	1	0	0	1	0
5	0	1	0	0	1
6	0	1	0	0	1
7	0	1	1	0	0
8	1	0	0	1	0
9	1	0	0	1	0
10	1	0	0	0	1
11	1	0	0	0	1

图4-7位图索引

- 图4-8对应了在为Gender和Country属性创建位图索引后执行查询时位图索引的计算过程：

  SELECT count(*) FROM customer

  WHERE Gender=’M’ AND Country in (’Mexico’ ,’USA’);

- 查询中的谓词条件对应已建立位图索引的属性，将谓词条件Gender=’M’转换为访问Gender值为’ M’的位图，查询条件Country in (’Mexico’ ,’USA’)转换为访问Country值为’ Mexico’和’ USA’的位图，并对两个位图执行OR操作。查询中的复合谓词条件转换为图4-8所示的多个位图之间的逻辑运算，位图运算结果对应的位图则指标了满足查询谓词条件的记录在表中的位置。

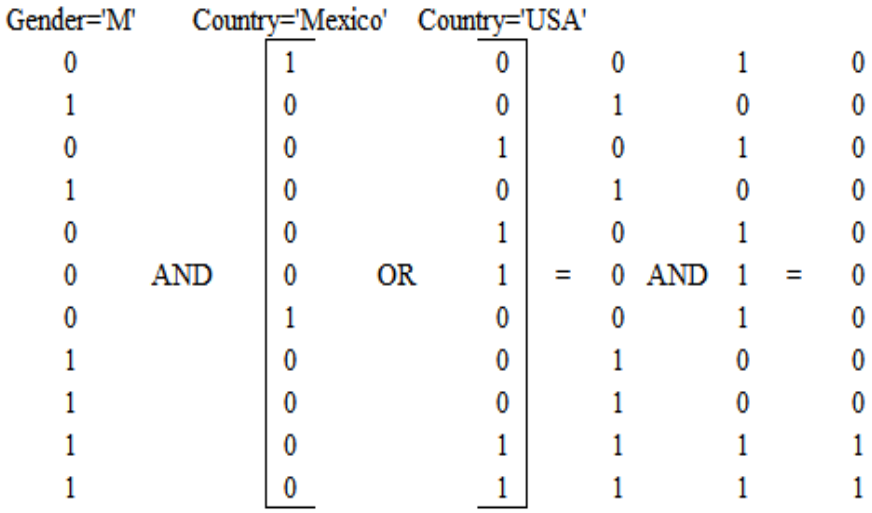


图4-8位图索引的使用

- 属性中的成员数量越多，位图索引中的位图数量越多，位图存储空间代价越大。但属性中的成员数量越多，每个位图中1的数量更加稀疏，位图运算对应的选择率越低，查询性能提升越大。当表中记录数量非常大时，稀疏的位图可以通过压缩技术缩减位图索引存储空间，同时，当位图很大时，位图运算也消耗大量的CPU计算资源，可以通过SIMD并行计算技术提高位图计算性能，也可以通过协处理器，如GPGPU、Phi协处理器所支持的512位SIMD计算能力来提高位图索引的计算性能。

第2节 数据定义SQL

- 图4-9中维表Customer的属性Gender和Country为低势集属性，事实表Sales中包含Customer表的外键属性customer_id。我们可以使用如下的SQL命令为Customer表的Gender和Country属性创建与Sales表的位图连接索引。

```
CREATE BITMAP INDEX sales_c_gender_country
ON sales(customer. gender, customer. country)
FROM sales, customer
WHERE sales.customer_id = customer.customer_id;
```

Customer

Customer_id	Gender	Country
1	F	Mexico
2	M	Canada
3	F	USA
4	M	Canada
5	F	USA

Bitmap Join index	Gender		Country		
sales_id	M	F	Mexico	Canada	USA
1	1	0	0	1	0
2	0	1	0	0	1
3	0	1	1	0	0
4	0	1	0	0	1
5	1	0	0	1	0
6	1	0	0	1	0
7	0	1	0	0	1
8	1	0	0	1	0
9	0	1	0	0	1
10	0	1	1	0	0
11	0	1	0	0	1

Sales

time_id	customer_id	Revenue
20120301	4	3452
20120301	3	4432
20120302	1	5356
20120303	5	2352
20120303	2	5536
20120304	4	6737
20120305	3	5648
20120306	2	9345
20120306	5	5547
20120307	1	7578
20120308	3	5533

图4-9位图连接索引的使用

- 位图索引表示的是属性成员在当前表中的位置信息，而位图连接索引则表示属性成员在连接表中的位置信息。位图连接索引相当于为物化连接表属性创建的位图索引，在实际应用中能够有效地减少连接操作代价。
- 位图索引和位图连接索引都是在选定属性上为所有成员创建位图，低势集的属性上创建的位图数量较少，索引空间开销较小，但由于属性成员数量少，每一个位图对应的选择率较高，对连接操作的加速能力较低，而高势集属性的成员数量较多，需要创建较多的位图，索引存储空间开销较大，但位图的选择率低，对连接操作的加速能力强，因此位图连接索引的创建和使用需要权衡位图连接索引的存储开销和查询性能优化收益而综合评估。

（7）存储索引Storage Index

- 存储索引是一种根据数据块元信息过滤查询访问数据块的索引技术。如图4-10所示，数据库为每设定大小的数据块在内存中建立汇总元信息，如最小值、最大值、数据块中记录数、数据型列累加和等信息，可以对常用列自动收集这些汇总信息并在内存建立存储索引。在查询执行时，首先扫描内存中存储索引各数据块的元信息，根据查询条件与元信息过滤掉不符合条件的数据块，如查询条件“PROD_CODE BETWEEN 75000 AND 90000”超出第一个数据块中PROD_CODE最大值，因此查询时可以完全跳过对该数据块的访问，查询条件与第二个数据块的最小值（39023）最大值（87431）范围有交集，因此需要扫描第二个数据块。存储索引只记录数据块中汇总的元信息，数据量极小，可以常驻内存。通过存储索引过滤掉与查询条件不相关的磁盘数据块，提高查询的I/O效率。当数据分布比较偏斜时，块中最小值与最大值分布也比较偏斜，在查询中存储索引的过滤效果较好。
- 与其他索引不同，存储索引不是一种精确的索引，而是基于元数据统计的粗糙过滤索引，索引访问的粒度是数据块，数据块内还需要通过扫描操作完成查询。

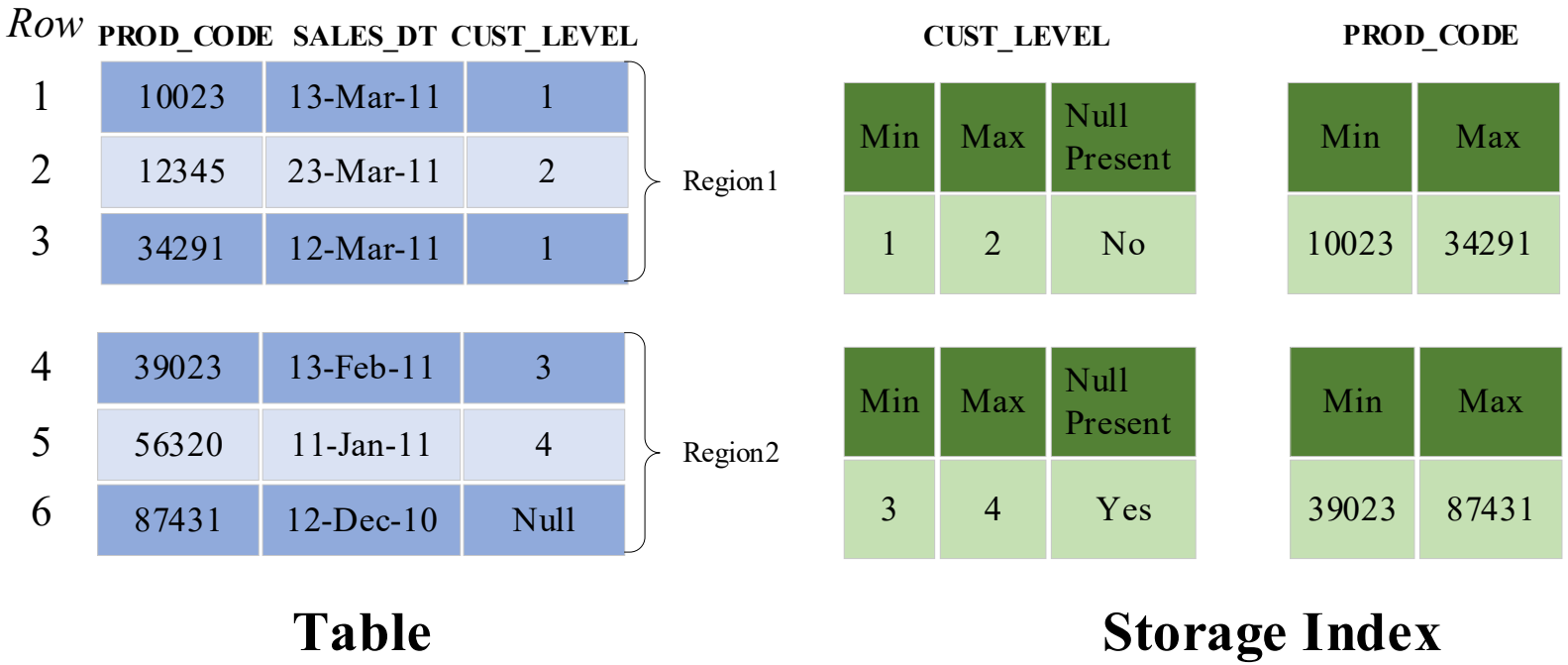

Row
PROD_CODE
SALES_DT
CUST_LEVEL
1
10023
13-Mar-11
1
2
12345
23-Mar-11
2
3
34291
12-Mar-11
1
Region1
4
39023
13-Feb-11
3
5
56320
11-Jan-11
4
6
87431
12-Dec-10
Null
Region2
CUST_LEVEL
Min
Max
Null Present
1
2
No
3
4
Yes
PROD_CODE
Min
Max
10023
34291
39023
87431
Table
Storage Index

图4-10存储索引

（8）列存储索引Column store index

- 列存储索引是一种基于列存储模型的索引结构。SQL Server 2012/2016/2017中采用了列存储索引技术，如图4-11所示，行记录以1M行为单位划分为row group，每个row group中的属性按列存储并进行压缩，采用字典表压缩技术的列需要在row group中存储字典表。在查询处理时，索引涉及的列在列存储索引上按列处理，提高查询处理性能。

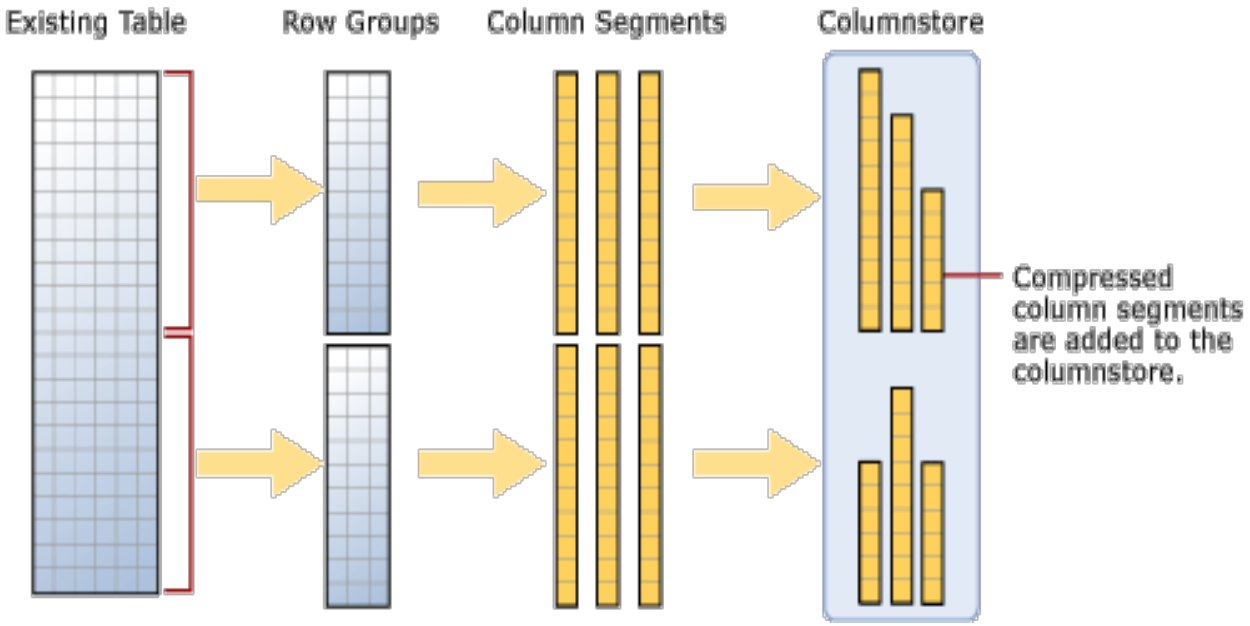


图4-11 列存储索引

第2节 数据定义SQL

- 在传统的磁盘行存储数据库中，当按指定键值查找时需要扫描全部的记录才能找到满足条件的记录，在查找过程中只有指定属性值被查找所使用，数据访问效率低。索引相对于将查找属性单独存储并通过索引优化查找操作，无论是在数据访问效率还是查找效率都有较好的性能。索引是数据库重要的查询优化技术之一，为频繁访问的列创建索引是典型的优化策略。但另一方面，索引需要额外的存储开销和管理代价，当数据更新时索引需要同步更新或者重建，增加数据库的更新时的代价，即索引优化了查找性能但恶化了更新性能，因此索引的使用需要综合权衡数据库的负载特点和应用特点，选择适当的索引策略以提高数据库的整体性能。

四、索引的创建与删除

- 索引是依赖于基本表的数据结构，为基本表中的索引列创建适合查找的索引结构，查询时先在索引上查找，然后再通过索引中记录的地址定位到基本表中对应的记录，以加快查找速度。
- 命令：CREATE INDEX
- 功能：创建索引。
- 语法：

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX index_name
    ON <object> (column_name [ ASC | DESC ] [ ,...n ] );
```

- SQL命令描述：
  - 命令为数据库对象object（表或视图）创建索引，索引可以建立在一列或多个列上，各列通过逗号分隔，ASC表示升序，DESC表示降序。
  - UNIQUE表示索引的每一个索引值对应唯一的数据记录。
  - CLUSTERED表示建立的索引是聚集索引。聚集索引按索引列（或多个列）值的顺序组织数据在表中的物理存储顺序，一个基本表上只能创建一个聚集索引，一些数据库默认为主键创建聚集索引。

– 索引需要占用额外的存储空间，基本表更新时索引也需要同步进行更新，数据库管理员需要权衡索引优化策略，有选择创建索引来达到以较低的存储和更新代价达到加速查询性能的目的。索引建立后，数据库系统在执行查询时会自动选择适合的索引作为查询存取路径，不需要用户指定索引的使用。

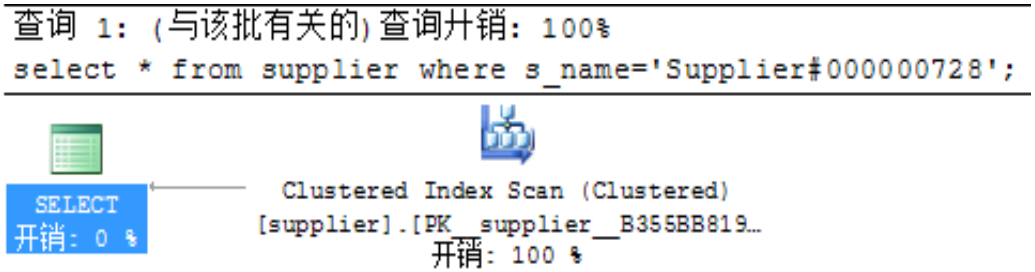
- SQL命令示例：

【例4-8】为TPC-H数据库的supplier表的s_name列创建唯一索引，为s_nation列和s_city列创建复合索引，其中s_nation为升序，s_city为降序。

```
CREATE UNIQUE INDEX s_name_Inx ON supplier(s_name);
CREATE INDEX s_n_c_Inx ON supplier(s_nationkey ASC, s_phone DESC);
```

- 在创建唯一索引时，要求唯一索引的列中不能有重复值，即唯一索引列应该是候选码。
- 查询：select * from supplier where s_name='Supplier#000000728';
- 在创建索引之前和之后执行时的执行计划不同，无索引时采用顺序扫描查找，建立索引后在索引列上的查找先在索引中进行查找，然后再从原始表中定位索引中查找到的记录。

第2节 数据定义SQL



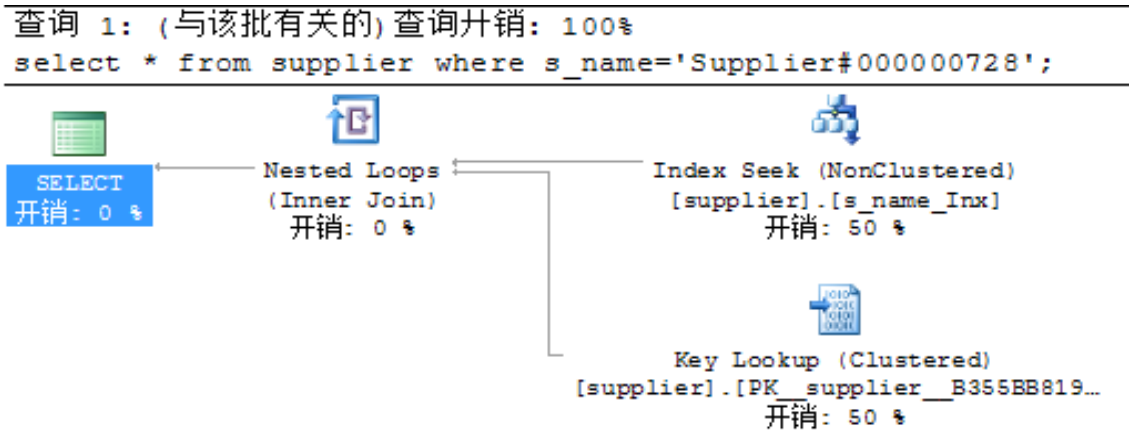


图4-12扫描与索引扫描

第2节 数据定义SQL

【例4-9】为TPC-H数据库的LINEITEM表创建列存储索引。

- 测试查询为简化的Q1查询，涉及LINEITEM表上多个列的谓词、分组、聚集计算，

```
SELECT l_returnflag, l_linestatus, SUM(l_extendedprice*(1-l_discount)*(1+l_tax))
FROM lineitem
WHERE l_shipdate <= '1998-12-01'
GROUP BY l_returnflag, l_linestatus;
```

- 查询执行的主要代价是在LINEITEM表上的扫描代价，通过全表扫描逐行读取查询相关属性，并完成查询处理任务。

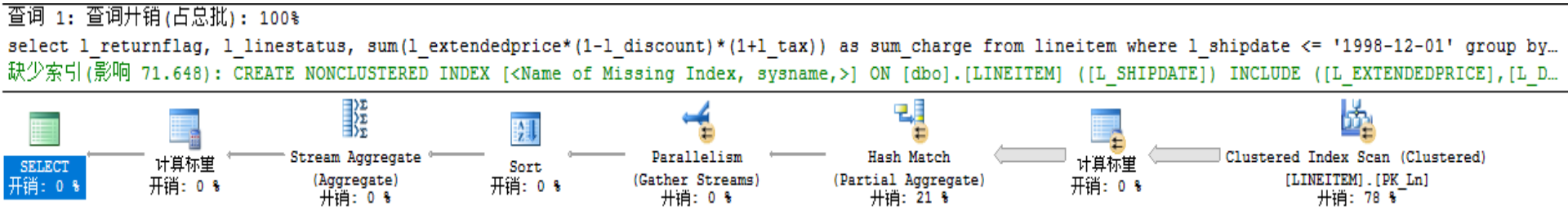


图4-13基于表扫描的查询计划

第2节 数据定义SQL

- 为查询中访问的列创建列存储索引，将查询的表扫描操作转换为高效的列存储索引扫描。创建列存储索引命令如下：

```
CREATE NONCLUSTERED COLUMNSTORE INDEX csindx_lineorder
ON lineitem(l_returnflag, l_linestatus, l_extendedprice,l_discount,l_tax,l_shipdate);
```

- 创建列存储索引后，执行查询时数据库查询处理引擎自动选择列存储索引加速查询处理性能，在查询计划中使用列存储索引扫描操作代替原始的表扫描操作，提高查询性能。

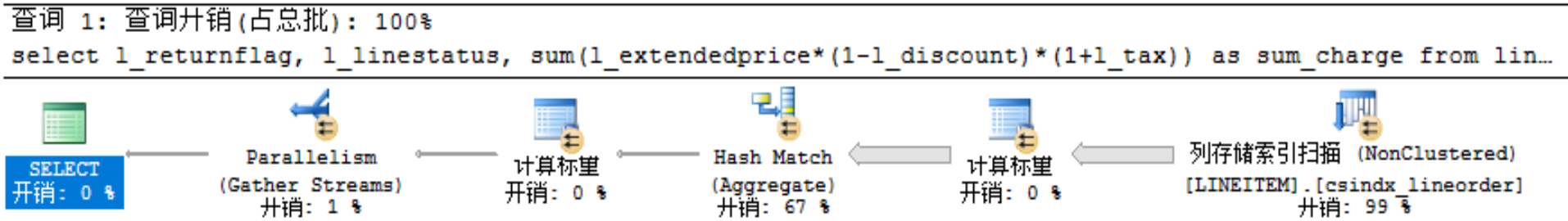


图4-14基于列存储索引扫描的查询计划

- 索引创建后由数据库系统自动使用和维护，不需要用户干预。索引提高了数据库查询性能，但数据的更新操作，如增、删、改操作在更新表中数据的同时也需要在相关索引中同步更新，当索引较多时降低了数据库更新操作性能。索引需要较大的存储空间，当索引数量较多时可能超过原始表大小，提高了数据库系统的维护成本。
- 命令：DROP INDEX
- 功能：删除索引。
- 语法：

  DROP INDEX [index_name ON <object>]

  [table_or_view_name.index_name];
- SQL命令描述：
  - 当索引过多或建立不当时，数据频繁的增、删、改会产生较多的索引维护代价，降低查询效率。用户可以删除不必要的索引来优化数据库性能。删除索引时索引名可以使用两种指定方法：index_name ON <object>和table_or_view_name.index_name，指示索引依赖的表名和索引名。
- SQL命令示例：

【例4-10】删除基本表supplier上的索引s_n_c_Inx和s_name_Inx。

```
DROP INDEX supplier.s_n_c_Inx;
DROP INDEX s_name_Inx ON supplier;
```

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3节 数据查询SQL

- 查询是数据库的核心操作，SQL提供SELECT语句进行数据查询。SELECT语句具有丰富的功能，在不同的数据库系统中语法各有不同，比较有代表性的格式如下所示：
- 命令：SELECT
- 功能：查询。
- 语法：

[WITH <common_table_expression>]
SELECT select_list [INTO new_table]
[FROM table_source] [WHERE search_condition]
[GROUP BY group_by_expression]
[HAVING search_condition]
[ORDER BY order_expression [ASC | DESC]]

第3节 数据查询SQL

- SQL命令描述：
  - WITH语句可以定义一个公用表表达式，将一个简单查询表达式定义为临时表使用。
  - SELECT语句的含义是从FROM子句table_source指定的基本表、视图、派生表或公用表表达式中按WHERE子句search_condition指定的条件表达式选择出目标列表达式select_list指定的元组属性，按GROUP BY子句group_by_expression指定的分组列进行分组，并按select_list指定的聚集函数进行聚集计算，分组聚集计算的结果按HAVING子句search_condition指定的条件输出，输出的结果按ORDER BY子句order_expression指定的列进行排序。
  - SELECT命令可以是单表查询，也可以是多表查询和嵌套查询，也可以通过集合操作将多个查询的结果组合，也可以在查询中使用派生表进行查询，下面分别对不同的查询执行方式进行分析和说明。

第3节 数据查询SQL

一、单表查询

- 单表查询是针对一个表的查询操作，主要包括选择、投影、聚集、分组、排序等关系操作，其中FROM子句指定查询的表名。

1.投影操作

- 选择输出表中全部或部分指定的列。

（1）查询全部的列

- 查询全部的列时，select_list可以用*或表中全部列名来表示。
- SQL命令示例：（示例表为TPCH.PART）

【例4 11】查询PART表中全部的记录。

```
SELECT * FROM PART;
```

或

```
SELECT P_PARTKEY, P_NAME, P_MFGR, P_BRAND, P_TYPE,
P_SIZE, P_CONTAINER, P_RETAILPRICE, P_COMMENT
FROM PART;
```

- 当表中列数量较多时，*能够更加快捷地指代全部的列。

（2）查询指定的列

- 通过在select_list中指定输出列的名称和顺序定义查询输出的列。

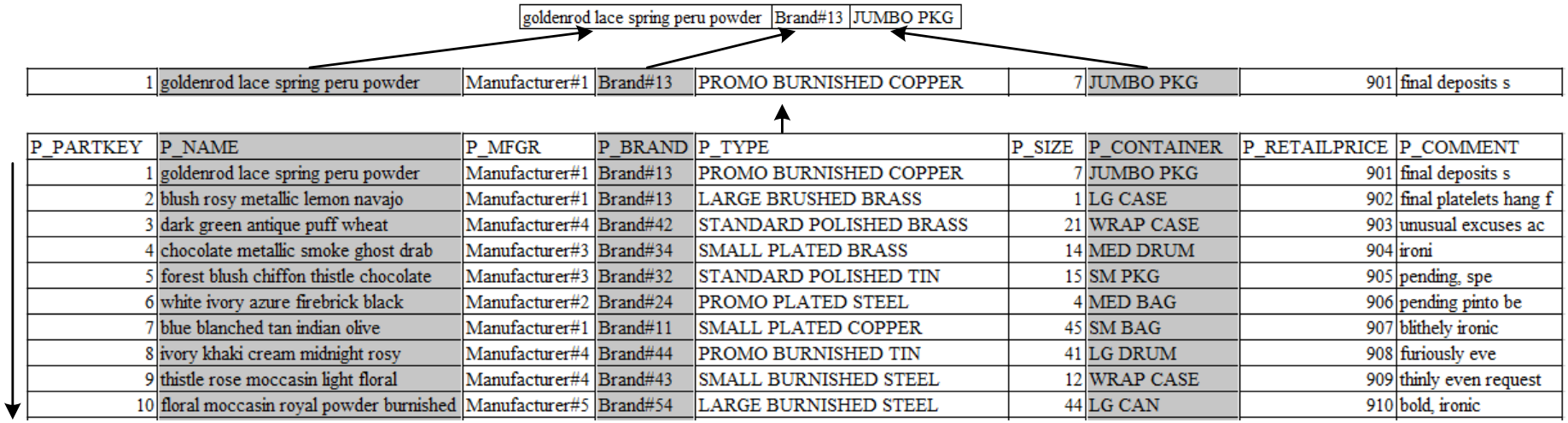
【例4 12】查询PART表中P_NAME、P_BRAND和P_CONTAINER列。

```
SELECT P_NAME, P_BRAND, P_CONTAINER FROM PART;
```

- 在查询执行时，从表PART中取出一个元组，按select_list中指定输出列的名称和顺序取出属性P_NAME, P_BRAND 和P_CONTAINER的值，组成一个新的元组输出。列输出的顺序可以与表中列存储的顺序不一致。

第3节 数据查询SQL

- 在行存储数据库中，各个列的属性顺序地存储在一起，虽然查询中可能只输出少数的列，但需要访问全部的元组才能投影出指定的少数列，如图4-15（A）所示，投影操作不能减少从磁盘访问数据的代价。



1	goldenrod lace spring peru powder	Manufacturer#1	Brand#13	PROMO BURNISHED COPPER	7	JUMBO PKG	901	final deposits s

P_PARTKEY	P_NAME	P_MFGR	P_BRAND	P_TYPE	P_SIZE	P_CONTAINER	P_RETAILPRICE	P_COMMENT
1	goldenrod lace spring peru powder	Manufacturer#1	Brand#13	PROMO BURNISHED COPPER	7	JUMBO PKG	901	final deposits s
2	blush rosy metallic lemon navajo	Manufacturer#1	Brand#13	LARGE BRUSHED BRASS	1	LG CASE	902	final platelets hang f
3	dark green antique puff wheat	Manufacturer#4	Brand#42	STANDARD POLISHED BRASS	21	WRAP CASE	903	unusual excuses ac
4	chocolate metallic smoke ghost drab	Manufacturer#3	Brand#34	SMALL PLATED BRASS	14	MED DRUM	904	ironi
5	forest blush chiffon thistle chocolate	Manufacturer#3	Brand#32	STANDARD POLISHED TIN	15	SM PKG	905	pending, spe
6	white ivory azure firebrick black	Manufacturer#2	Brand#24	PROMO PLATED STEEL	4	MED BAG	906	pending pinto be
7	blue blanched tan indian olive	Manufacturer#1	Brand#11	SMALL PLATED COPPER	45	SM BAG	907	blithely ironic
8	ivory khaki cream midnight rosy	Manufacturer#4	Brand#44	PROMO BURNISHED TIN	41	LG DRUM	908	furiously eve
9	thistle rose moccasin light floral	Manufacturer#4	Brand#43	SMALL BURNISHED STEEL	12	WRAP CASE	909	thinly even request
10	floral moccasin royal powder burnished	Manufacturer#5	Brand#54	LARGE BURNISHED STEEL	44	LG CAN	910	bold, ironic

（A）行存储时的投影操作

图4-15行存储和列存储投影操作的数据访问方式

第3节 数据查询SQL

- 而列存储是将各列独立存储，查询可以只读取查询访问的列，如图4-15（B）所示，数据的磁盘访问效率更高。

goldenrod lace spring peru powder　　Brand#13　　JUMBO PKG

P_PARTKEY	P_NAME	P_MFGR	P_BRAND	P_TYPE	P_SIZE	P_CONTAINER	P_RETAILPRICE	P_COMMENT
1	goldenrod lace spring peru powder	Manufacturer#1	Brand#13	PROMO BURNISHED COPPER	7	JUMBO PKG	901	final deposits s
2	blush rosy metallic lemon navajo	Manufacturer#1	Brand#13	LARGE BRUSHED BRASS	1	LG CASE	902	final platelets hang f
3	dark green antique puff wheat	Manufacturer#4	Brand#42	STANDARD POLISHED BRASS	21	WRAP CASE	903	unusual excuses ac
4	chocolate metallic smoke ghost drab	Manufacturer#3	Brand#34	SMALL PLATED BRASS	14	MED DRUM	904	ironi
5	forest blush chiffon thistle chocolate	Manufacturer#3	Brand#32	STANDARD POLISHED TIN	15	SM PKG	905	pending, spe
6	white ivory azure firebrick black	Manufacturer#2	Brand#24	PROMO PLATED STEEL	4	MED BAG	906	pending pinto be
7	blue blanched tan indian olive	Manufacturer#1	Brand#11	SMALL PLATED COPPER	45	SM BAG	907	blithely ironic
8	ivory khaki cream midnight rosy	Manufacturer#4	Brand#44	PROMO BURNISHED TIN	41	LG DRUM	908	furiously eve
9	thistle rose moccasin light floral	Manufacturer#4	Brand#43	SMALL BURNISHED STEEL	12	WRAP CASE	909	thinly even request
10	floral moccasin royal powder burnished	Manufacturer#5	Brand#54	LARGE BURNISHED STEEL	44	LG CAN	910	bold, ironic

（B）列存储时的投影操作

图4-15行存储和列存储投影操作的数据访问方式

（3）查询表达式列

- SELECT子句中的目标列表达式既可以是表中的列，也可以是列表达式，表达式可以是列的算术/字符串表达式、字符串常量、函数等，可以灵活地输出原始列或派生列。
- SQL命令示例：

计算一元线性回归方程系数

【例4-13】查询LINEITEM表中COMMITDATE、RECEIPTDATE、间隔时间、折扣后价格以及折扣及税后价格。

```
Select L_COMMITDATE, L_RECEIPTDATE, 'Interval Days:' as Receipting,
DATEDIFF (DAY, L_COMMITDATE, L_RECEIPTDATE) as IntervalDay,
L_EXTENDEDPRICE*(1-L_DISCOUNT) as DiscountedPrice,
L_EXTENDEDPRICE*(1-L_DISCOUNT)*(1+L_TAX) as DiscountedTaxedPrice
from LINEITEM;
```

– SQL查询解析：输出表中原始的列信息，常量'Interval Days:'作为常量列，as短语为列设置别名，日期函数DATEDIFF (DAY, L_COMMITDATE, L_RECEIPTDATE)计算RECEIPTDATE与COMMITDATE间隔的天数并作为IntervalDay输出，折扣价格表达式L_EXTENDEDPRICE*(1-L_DISCOUNT)与折扣税后价格表达式L_EXTENDEDPRICE*(1-L_DISCOUNT)*(1+L_TAX)结果作为新列输出。

第3节 数据查询SQL

	L_COMMITDATE	L_RECEIPTDATE	Receipting	IntevalDay	DiscountedPrice	DiscountedTaxedPrice
1	1996-02-12	1996-03-22	Inteval Days:	39	21168	21168
2	1996-02-28	1996-04-20	Inteval Days:	52	45983	45983
3	1996-03-05	1996-01-31	Inteval Days:	-34	13309	13309
4	1996-03-30	1996-05-16	Inteval Days:	47	28955	28955
5	1996-03-14	1996-04-01	Inteval Days:	18	22824	22824
6	1996-02-07	1996-02-03	Inteval Days:	-4	49620	49620
7	1997-01-14	1997-02-02	Inteval Days:	19	44694	44694
8	1994-01-04	1994-02-23	Inteval Days:	50	54058	54058
9	1993-12-20	1993-11-24	Inteval Days:	-26	46796	46796
10	1993-11-22	1994-01-23	Inteval Days:	62	39890	39890

图4-16查询表达式列输出

- 如图4-16所示，列表达式在查询时实时生成列表达式结果并输出，扩展了表中数据的应用范围，增加了查询的灵活性。

第3节 数据查询SQL

（4）投影出列中不同的成员

- 列中取值既可以各不相同，也允许存在重复值。对于候选码属性，列中的取值必须各不相同，在此基础上才能建立唯一索引或主键索引。非码属性中存在重复值，通过DISTINCT命令可以输出指定列中不重复取值的成员。
- SQL命令示例：

【例4-14】查出LINEITEM表中各订单项的L_SHIPMODE方式以及查询共有哪些L_SHIPMODE方式。

select L_SHIPMODE from LINEITEM;

– SQL命令解析：输出L_SHIPMODE列中全部的取值，包括了重复的取值。

select distinct L_SHIPMODE from LINEITEM;

– SQL命令解析：通过DISTINCT短语指定列L_SHIPMODE只输出不同取值的成员，列中的每个取值只输出一次。

- 码属性上的DISTINCT成员数量与表中记录行数相同，非码属性上的DISTINCT成员数量小于等于表中记录行数。通过对列DISTINCT取值的分析，用户可以了解数据的分布特征。

	L_SHIPMODE
1	TRUCK
2	MAIL
3	REG AIR
4	AIR
5	FOB
6	MAIL
7	RAIL
8	AIR
9	RAIL
10	SHIP

	L_SHIPMODE
1	SHIP
2	RAIL
3	AIR
4	FOB
5	TRUCK
6	MAIL
7	REG AIR

图4-17投影操作与投影去重操作

2.选择操作

- 选择操作是通过WHERE子句的条件表达式对表中记录进行筛选，输出查询结果。常用的条件表达式可以分为6类，如下表所示：

查询条件	查询条件运算符
比较大小	=, >, <, >=, <=, !=, <>, NOT+比较运算符
范围判断	BETWEEN AND , NOT BETWEEN AND
集合判断	IN, NOT IN
字符匹配	LIKE, NOT LIKE
空值判断	IS NULL, IS NOT NULL
逻辑运算	AND, OR, NOT

（1）比较大小

- 比较运算符对应具有大小关系的数值型、字符型、日期型等数据上的比较操作，通常是列名+比较操作符+常量或变量的格式，在实际应用中可以与包括函数的表达式共同使用。
- SQL命令示例：

【例4-15】输出LINEITEM表中满足条件的记录。

```
select * from LINEITEM where L_QUANTITY>45;
```

– SQL命令解析：输出LINEITEM表中L_QUANTITY大于45的记录。

```
select * from LINEITEM where L_SHIPINSTRUCT='COLLECT COD';
```

– SQL命令解析：输出表中L_SHIPINSTRUCT值为COLLECT COD的记录。

```
select * from LINEITEM where NOT L_COMMITDATE>L_SHIPDATE;
```

– SQL命令解析：输出表中L_COMMITDATE时间不晚于L_SHIPDATE时间的记录。

```
select * from LINEITEM where
DATEDIFF(DAY,L_COMMITDATE,L_RECEIPTDATE)> 10;
```

– SQL命令解析：输出表中RECEIPTDATE与COMMITDATE超过10天的记录。

（2）范围判断

- 范围操作符BETWEEN AND 和NOT BETWEEN AND 用于判断元组条件表达式是否在或不在指定范围之内。C BETWEEN a AND b等价于C >=a AND C <= b。
- SQL命令示例：

【例4-16】输出LINEITEM表中指定范围之间的记录。

```
select * from LINEITEM
where L_COMMITDATE between L_SHIPDATE and L_RECEIPTDATE;
```

– SQL命令解析：输出LINEITEM表中COMMITDATE介于SHIPDATE和RECEIPTDATE之间的记录。

```
select * from LINEITEM
where L_COMMITDATE not between '1996-01-01' and '1997-12-31';
```

– SQL命令解析：输出LINEITEM表中1996至1997年之外的记录。

日期函数*

- DATEADD(*datepart*,*number*,*date*)
  - *date* 参数是合法的日期表达式。*number* 是您希望添加的间隔数；对于未来的时间，此数是正数，对于过去的时间，此数是负数。
- DATEDIFF(*datepart*,*startdate*,*enddate*)
  - DATEDIFF() 函数返回两个日期之间的时间。
  - *startdate* 和 *enddate* 参数是合法的日期表达式。
- 示例：
  - select
  - l_shipdate,
  - DATEADD(YY,5,l_shipdate) as date_5Year_after,
  - DATEADD(Q,5,l_shipdate) as date_5Quarter_after,
  - DATEADD(WW,5,l_shipdate) as date_5Week_after,
  - DATEADD(DD,-25,l_shipdate) as date_25D_before,
  - DATEDIFF(D,l_shipdate,l_receiptdate) as daygap
  - from LINEITEM;

	l_shipdate	date_5Year_after	date_5Quarter_after	date_5Week_after	date_25D_before	daygap
1	1998-02-05	2003-02-05	1999-05-05	1998-03-12	1998-01-11	9
2	1997-12-26	2002-12-26	1999-03-26	1998-01-30	1997-12-01	18
3	1997-11-25	2002-11-25	1999-02-25	1997-12-30	1997-10-31	7
4	1998-02-14	2003-02-14	1999-05-14	1998-03-21	1998-01-20	13
5	1998-01-21	2003-01-21	1999-04-21	1998-02-25	1997-12-27	16
6	1996-04-07	2001-04-07	1997-07-07	1996-05-12	1996-03-13	27

datepart	缩写
年	yy, yyyy
季度	qq, q
月	mm, m
年中的日	dy, y
日	dd, d
周	wk, ww
星期	dw, w
小时	hh
分钟	mi, n
秒	ss, s
毫秒	ms
微妙	mcs
纳秒	ns

（3）集合判断

- 集合判断操作符IN和NOT IN用于判断表达式是否在指定集合范围之内。集合判断操作符C IN (a,b,c)等价于C= a OR C=b OR C=c。集合操作符中使用时更加简洁方便。
- SQL命令示例：

【例4 17】输出LINEITEM表中集合之内的记录。

```
select * from LINEITEM where L_SHIPMODE in ('MAIL', 'SHIP');
```

– SQL命令解析：输出L_SHIPMODE类型为MAIL和SHIP的记录。

```
select * from PART where P_SIZE not in (49,14,23,45,19,3,36,9);
```

– SQL命令解析：输出PART表中为P_SIZE不是49,14,23,45,19,3,36,9的记录。

– 当条件列为不同的数据类型时，IN集合中常量的数据类型应该与查询列数据类型格式保持一致。

（4）字符匹配

- 字符匹配操作符用于字符型数据上的模糊查询，其语法格式为：

  match_expression [NOT] LIKE pattern [ESCAPE escape_character]

  – match_expression为需要匹配的字符表达式。 pattern为匹配字符串，可以是完整的字符串，也可以是包含通配符%和_的字符串，其中：

    % 表示任意长度的字符串。

    _ 表示任意单个字符。

  – ESCAPE escape_character表示escape_character为换码字符，换码符后面的字符为普通字符。

- SQL命令示例：

【例4-18】输出模糊查询的结果。

select * from PART where P_TYPE like 'PROMO%';

– SQL命令解析：输出PART表中P_TYPE列中以PROMO开头的记录。

select * from SUPPLIER where S_COMMENT like '%Customer%Complaints%';

– SQL命令解析：输出SUPPLIER表中S_COMMENT中任意位置包含Customer并且后面字符中包含Complaints的记录。

select * from PART where P_CONTAINER like '% _AG';

– SQL命令解析：输出PART表P_CONTAINER列中倒数第3个为任意字符，最后2个字符为AG的记录。

select * from LINEITEM where L_COMMENT like '%return rate __\%for%' ESCAPE '%';

– SQL命令解析：输出LINEITEM表L_COMMENT列中包含return rate 和两位数字、百分比符号和for字符的记录，其中_为通配符，由'\'表示其后的%为百分比符号。

（5）空值判断

- 在数据库中，空值一般表示数据未知、不适用或将在以后添加数据。空值不同于空白或零值，空值用NULL表示，在查询中判断空值时，需要在 WHERE 子句中使用 IS NULL 或 IS NOT NULL，不能使用=NULL。
- SQL命令示例：

【例4-19】输出LINEITEM表中没有客户评价L_COMMENT的记录。

```
select * from LINEITEM where L_COMMENT is NULL;
```

– SQL命令解析：输出LINEITEM表中L_COMMENT列为空值的记录。

（6）复合条件表达式

- 逻辑运算符AND和OR可以连接多个查询条件，实现在表上按照多个条件表达式的复合条件进行查询。AND的优化级高于OR，可以通过括号改变逻辑运算符的优化级。
- SQL命令示例：
- 【例4-20】输出LINEITEM表中满足复合条件的记录。

```
select sum(L_EXTENDEDPRICE*L_DISCOUNT) as revenue from LINEITEM
where L_SHIPDATE between '1994-01-01' and '1994-12-31'
and L_DISCOUNT between 0.06 - 0.01 and 0.06 + 0.01 and L_QUANTITY < 24;
```

第3节 数据查询SQL

- SQL命令解析：输出LINEITEM表中SHIPDATE在1994年、折扣在5%-7%之间、数量小于24的订单项记录。多个查询条件用AND、OR连接，AND优先级高于OR。

```
select * from LINEITEM
where L_SHIPMODE in ('AIR', 'AIR REG')
and L_SHIPINSTRUCT ='DELIVER IN PERSON'
and ((L_QUANTITY >= 10 and L_QUANTITY <= 20) or (L_QUANTITY >= 30 and L_QUANTITY <= 40));
```

- SQL命令解析：输出LINEITEM表中SHIPMODE列为AIR或AIR REG，SHIPINSTRUCT类型为DELIVER IN PERSON，QUANTITY在10与20之间或30与40之间的记录。

- 当查询条件中包含多个由AND和OR连接的表达式时，需要适当地使用括号保证复合查询条件执行顺序的正确性。

第3节 数据查询SQL

- 3.聚集操作
- 选择和投影操作查询对应的是元组操作，查看的是记录的明细。数据库的聚集函数提供了对列中数据总量的统计方法，为用户提供对数据总量的计算方法。SQL提供的聚集函数主要包括：

COUNT(*)	统计元组的个数
COUNT([DISTINCT|ALL]<column_name>)	统计一列中不同值的个数
SUM([DISTINCT|ALL]< expression >)	计算表达式的总和
AVG([DISTINCT|ALL]< expression >)	计算表达式的平均值
MAX([DISTINCT|ALL]< expression >)	计算表达式的最大值
MIN([DISTINCT|ALL]< expression >)	计算表达式的最小值

- 当指定DISTINCT短语时，聚集计算时只计算列中不重复值记录，缺省（ALL）时聚集计算对列中所有的值进行计算。COUNT(*)为统计表中元组的数量，COUNT指定列则统计该列中非空元组的数量。聚集计算的对象可以是表中的列，也可以是包含函数的表达式。

第3节 数据查询SQL

- SQL命令示例：

【例4-21】执行TPC-H查询Q1中聚集计算部分。

```
select
sum(L_QUANTITY) as sum_qty,
sum(L_EXTENDEDPRICE) as sum_base_price,
sum(L_EXTENDEDPRICE*(1-L_DISCOUNT)) as sum_disc_price,
sum(L_EXTENDEDPRICE*(1-L_DISCOUNT)*(1+L_TAX)) as sum_charge,
avg(L_QUANTITY) as avg_qty,
avg(L_EXTENDEDPRICE) as avg_price,
avg(L_DISCOUNT) as avg_disc,
count(*) as count_order
from LINEITEM
```

- SQL命令分析：统计LINEITEM表中不同表达式的聚集计算结果。COUNT对象是*时表示统计表中记录数量，聚集函数可以对原始列或表达式进行聚集计算，均值AVG函数为导出函数，通过SUM与COUNT聚集结果计算而得到均值。

【例4-22】统计LINEITEM表中L_QUANTITY列的数据特征。

```
select count(distinct L_QUANTITY) as CARD, max(L_QUANTITY) as MaxValue,
min(L_QUANTITY) as MinValue from LINEITEM;
```

- SQL命令分析：统计LINEITEM表中L_QUANTITY列中不同取值的数量，最大值与最小值。

【例4-23】统计ORDERS表中高优化级与低优化级订单的数量。

```
select sum(case
when O_ORDERPRIORITY ='1-URGENT' or O_ORDERPRIORITY ='2-HIGH'
then 1 else 0 end) as high_line_count,
sum(case
when O_ORDERPRIORITY <> '1-URGENT' and O_ORDERPRIORITY <> '2-HIGH'
then 1 else 0 end) as low_line_count
from ORDERS;
```

- SQL命令分析：通过case语句根据构建的选择条件输出分支结果，并对结果进行聚集计算。

4.分组操作

- GPOUP BY语句将查询记录集按指定的一列或多列进行分组，然后对相同分组的记录进行聚集计算。分组操作扩展了聚集函数的应用范围，将一个汇总结果细分为若干个分组上的聚集计算结果，为用户提供更多维度、更细粒度的分析结果。
- SQL命令示例：

【例4-24】对LINEITEM表按RETURNFLAG,SHIPMODE不同的方式统计销售数量。

```
select sum(L_QUANTITY) as sum_quantity from LINEITEM;
```

– SQL命令解析：统计LINEITEM表所有记录L_QUANTITY的汇总值。

```
select L_RETURNFLAG,sum(L_QUANTITY) as sum_quantity from LINEITEM
group by L_RETURNFLAG;
```

– SQL命令解析：按L_RETURNFLAG属性分组统计LINEITEM表所有记录L_QUANTITY的汇总值。

```
select L_RETURNFLAG,L_LINESTATUS,sum(L_QUANTITY) as sum_quantity
from LINEITEM group by L_RETURNFLAG,L_LINESTATUS;
```

– SQL命令解析：按L_RETURNFLAG和L_LINESTATUS属性分组统计LINEITEM表所有记录L_QUANTITY的汇总值。

第3节 数据查询SQL

- 下面给出了三个SQL命令按不同的粒度分组聚集计算的结果，为用户展示了一个分析维度由少到多，粒度由粗到细的聚集计算过程。

	sum_quantity
1	153078795

	L_RETURNFLAG	sum_quantity
1	R	37719753
2	N	77624935
3	A	37734107

	L_RETURNFLAG	L_LINESTATUS	sum_quantity
1	R	F	37719753
2	N	O	76633518
3	A	F	37734107
4	N	F	991417

图4-18不同粒度分组统计结果

第3节 数据查询SQL

- 在分析处理任务中，GROUP BY子句中的多个分组属性可以看作是多个聚合计算维度，如图4-19所示，三个分组属性{a,b,c}构成一个三维聚合计算空间，包含：23个聚合分组：{a,b,c}、{a,b}、{a, c}、{b,c}、{a}、{b}、{c}、{}，代表三个分组属性所构成的所有可能的分组方案。

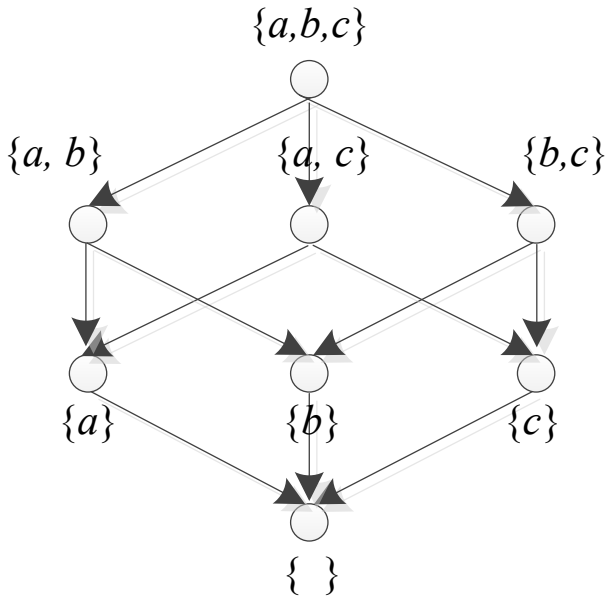


图4-19聚合维度

第3节 数据查询SQL

- SQL的GROUP BY子句中支持按照简单分组、上卷分组和CUBE分组方式进行聚合计算。

  GROUP BY

  – < group_by_expression >：直接按分组属性列表进行分组
  – | ROLLUP (<group_by_expression >)：按分组属性列表的上卷轴分组
  – | CUBE (<spec>)：按分组属性列表的数据立方体分组

- 例如：

  SELECT a, b, c, SUM (<expression>)

  FROM T

  GROUP BY ROLLUP (a,b,c);

  – 查询按 (a, b, c)、(a, b) 和 (a) 值的每个唯一组合生成一个带有小计的行。还将计算一个总计行。

- 查询：

  SELECT a, b, c, SUM (<expression>)

  FROM T

  GROUP BY CUBE (a,b,c);

  – 针对 < a,b,c > 中表达式的所有排列输出一个分组。生成的分组数等于 (2n)，其中 n = 分组子句中的表达式数。

第3节 数据查询SQL

【例4-25】 输出LINEITEM表中L_QUANTITY按L_RETURNFLAG、L_LINESTATUS、L_SHIPINSTRUCT三个属性的聚合结果。

（1）简单GROUP BY分组

```
select L_RETURNFLAG, L_LINESTATUS, L_SHIPINSTRUCT,
sum(L_QUANTITY) as sum_quantity
from LINEITEM
group by L_RETURNFLAG, L_LINESTATUS, L_SHIPINSTRUCT;
```

– SQL命令解析：查询按L_RETURNFLAG、L_LINESTATUS、L_SHIPINSTRUCT三个属性直接进行分组聚集计算，查询结果如图4-20所示。

	L_RETURNFLAG	L_LINESTATUS	L_SHIPINSTRUCT	sum_quantity
1	A	F	DELIVER IN PERSON	9434359
2	N	O	DELIVER IN PERSON	19149263
3	N	O	TAKE BACK RETURN	19177526
4	N	F	TAKE BACK RETURN	244383
5	N	F	NONE	251993
6	R	F	TAKE BACK RETURN	9415686
7	N	O	NONE	19171510
8	R	F	DELIVER IN PERSON	9430274
9	A	F	NONE	9438372
10	R	F	NONE	9414140
11	N	F	COLLECT COD	245385
12	N	O	COLLECT COD	19135219
13	N	F	DELIVER IN PERSON	249656
14	A	F	COLLECT COD	9432010
15	A	F	TAKE BACK RETURN	9429366
16	R	F	COLLECT COD	9459653

图4-20简单group by分组

第3节 数据查询SQL

（2）ROLLUP GROUP BY分组

```
select L_RETURNFLAG, L_LINESTATUS, L_SHIPINSTRUCT,
sum(L_QUANTITY) as sum_quantity
from LINEITEM
group by ROLLUP (L_RETURNFLAG, L_LINESTATUS, L_ SHIPINSTRUCT);
```

– SQL命令解析：查询以L_RETURNFLAG、L_LINESTATUS、L_SHIPINSTRUCT三个属性为基础，以L_RETURNFLAG为上卷轴由细到粗进行多个分组属性聚集计算。查询结果如图4-21所示。

	L_RETURNFLAG	L_LINESTATUS	L_SHIPINSTRUCT	sum_quantity
1	A	F	COLLECT COD	9432010
2	A	F	DELIVER IN PERSON	9434359
3	A	F	NONE	9438372
4	A	F	TAKE BACK RETURN	9429366
5	A	F	NULL	37734107
6	A	NULL	NULL	37734107
7	N	F	COLLECT COD	245385
8	N	F	DELIVER IN PERSON	249656
9	N	F	NONE	251993
10	N	F	TAKE BACK RETURN	244383
11	N	F	NULL	991417
12	N	O	COLLECT COD	19135219
13	N	O	DELIVER IN PERSON	19149263
14	N	O	NONE	19171510
15	N	O	TAKE BACK RETURN	19177526
16	N	O	NULL	76633518
17	N	NULL	NULL	77624935
18	R	F	COLLECT COD	9459653
19	R	F	DELIVER IN PERSON	9430274
20	R	F	NONE	9414140
21	R	F	TAKE BACK RETURN	9415686
22	R	F	NULL	37719753
23	R	NULL	NULL	37719753
24	NULL	NULL	NULL	153078795

（3）CUBE GROUP BY分组

```
select L_RETURNFLAG, L_LINESTATUS, L_SHIPINSTRUCT,
sum(L_QUANTITY) as sum_quantity
from LINEITEM
group by CUBE (L_RETURNFLAG, L_LINESTATUS, L_SHIPINSTRUCT);
```

- SQL命令解析：查询以L_RETURNFLAG 、 L_LINESTATUS和L_SHIPINSTRUCT三个属性为基础，为每一个分组属性组合进行分组聚集计算

第3节 数据查询SQL

	L_RETURNFLAG	L_LINESTATUS	L_SHIPINSTRUCT	sum_quantity
1	A	F	COLLECT COD	9432010
2	N	F	COLLECT COD	245385
3	R	F	COLLECT COD	9459653
4	NULL	F	COLLECT COD	19137048
5	N	O	COLLECT COD	19135219
6	NULL	O	COLLECT COD	19135219
7	NULL	NULL	COLLECT COD	38272267
8	A	F	DELIVER IN PERSON	9434359
9	N	F	DELIVER IN PERSON	249656
10	R	F	DELIVER IN PERSON	9430274
11	NULL	F	DELIVER IN PERSON	19114289
12	N	O	DELIVER IN PERSON	19149263
13	NULL	O	DELIVER IN PERSON	19149263
14	NULL	NULL	DELIVER IN PERSON	38263552
15	A	F	NONE	9438372
16	N	F	NONE	251993
17	R	F	NONE	9414140
18	NULL	F	NONE	19104505
19	N	O	NONE	19171510
20	NULL	O	NONE	19171510
21	NULL	NULL	NONE	38276015
22	A	F	TAKE BACK RETURN	9429366
23	N	F	TAKE BACK RETURN	244383
24	R	F	TAKE BACK RETURN	9415686
25	NULL	F	TAKE BACK RETURN	19089435
26	N	O	TAKE BACK RETURN	19177526
27	NULL	O	TAKE BACK RETURN	19177526
28	NULL	NULL	TAKE BACK RETURN	38266961
29	NULL	NULL	NULL	153078795
30	A	NULL	COLLECT COD	9432010
31	A	NULL	DELIVER IN PERSON	9434359
32	A	NULL	NONE	9438372
33	A	NULL	TAKE BACK RETURN	9429366
34	A	NULL	NULL	37734107
35	N	NULL	COLLECT COD	19380604
36	N	NULL	DELIVER IN PERSON	19398919
37	N	NULL	NONE	19423503
38	N	NULL	TAKE BACK RETURN	19421909
39	N	NULL	NULL	77624935
40	R	NULL	COLLECT COD	9459653
41	R	NULL	DELIVER IN PERSON	9430274
42	R	NULL	NONE	9414140
43	R	NULL	TAKE BACK RETURN	9415686
44	R	NULL	NULL	37719753
45	A	F	NULL	37734107
46	N	F	NULL	991417
47	R	F	NULL	37719753
48	NULL	F	NULL	76445277
49	N	O	NULL	76633518
50	NULL	O	NULL	76633518

图4-22 CUBE group by分组

第3节 数据查询SQL

SQL命令示例：

【例4-26】输出LINEITEM表订单中项目超过5项的订单号。

```
select L_ORDERKEY, count(*) as order_counter
from LINEITEM
group by L_ORDERKEY
having count(*)>=5;
```

– SQL命令解析：HAVING短语中的COUNT(*)>5作为分组聚集计算结果的过滤条件，对分组聚集结果进行筛选。

【例4-27】输出LINEITEM表订单中项目超过5项并且平均销售数量在28和30之间的订单的平均销售价格。

```
select L_ORDERKEY, avg(L_EXTENDEDPRICE)
from LINEITEM
group by L_ORDERKEY
having avg(L_QUANTITY) between 28 and 30 and count(*)>5;
```

– SQL命令解析：HAVING短语中可以使用输出目标列中没有的聚集函数表达式。如HAVING avg(L_QUANTITY) between 28 and 30 and count(*)>5短语中表达式avg(L_QUANTITY) between 28 and 30和count(*)>5均不是查询输出的聚集函数表达式，只用于对分组聚集计算结果进行筛选。

5.排序操作

- SQL中的ORDER BY子句用于对查询结果按照指定的属性顺序排列，排序属性可以是多个，DESC短语表示降序，默认为升序（ASC）。
- SQL命令示例：

【例4-28】对LINEITEM表进行分组聚集计算，输出排序的查询结果。

```
select L_RETURNFLAG,L_LINESTATUS,
sum(L_QUANTITY) as sum_quantity
from LINEITEM
group by L_RETURNFLAG,L_LINESTATUS
order by L_RETURNFLAG,L_LINESTATUS;
```

– SQL命令解析：对查询结果按分组属性排序，第一排序属性为L_RETURNFLAG，第二排序属性为L_LINESTATUS。

	L_RETURNFLAG	L_LINESTATUS	sum_quantity
1	A	F	37734107
2	N	F	991417
3	N	O	76633518
4	R	F	37719753

第3节 数据查询SQL


```
select L_RETURNFLAG,L_LINESTATUS,
sum(L_QUANTITY) as sum_quantity
from LINEITEM
group by L_RETURNFLAG,L_LINESTATUS
order by sum(L_QUANTITY) DESC;
```

– SQL命令解析：对分组聚集结果按聚集表达式结果降序排列。

	L_RETURNFLAG	L_LINESTATUS	sum_quantity
1	N	O	76633518
2	A	F	37734107
3	R	F	37719753
4	N	F	991417

第3节 数据查询SQL


```
select L_RETURNFLAG,L_LINESTATUS,
sum(L_QUANTITY) as sum_quantity
from LINEITEM
group by L_RETURNFLAG,L_LINESTATUS
order by sum_quantity DESC;
```

– SQL命令解析：当聚集表达式设置别名时，可以使用别名作为排序属性名，指代聚集表达式。

	L_RETURNFLAG	L_LINESTATUS	sum_quantity
1	N	O	76633518
2	A	F	37734107
3	R	F	37719753
4	N	F	991417

二、连接查询

- 连接操作通过连接表达式将两个以上的表连接起来进行查询处理。连接操作是数据库中最重要的关系操作，包括笛卡尔连接、等值连接、自然连接、非等值连接、自身连接、外连接和复合条件连接等不同的类型。

1.笛卡尔连接、等值连接、自然连接、非等值连接

- 在SQL命令中，当在FROM子句中指定了连接的表名，但没有设置连接条件时，两表执行笛卡尔连接，如：select * from NATION,REGION;NATION表中的每一条元组与REGION表中的全部元组进行连接。
- 当在SQL命令中进一步连接列的名称、以及连接列需要满足的连接条件（连接谓词）时，执行普通连接操作。连接操作中连接表名通常为FROM子句中的表名列表，连接条件为WHERE子句中的连接表达式，其格式为：

  [<table_name1>.]<column_name1> <operator>
  [<table_name2>.]<column_name2>

- 其中，比较运算符operator主要为=、>、<、>=、<=、!=(<>)等比较运算符。当比较运算符为=时称为等值连接，使用其他不等值运算符时的连接称为非等值连接。

第3节 数据查询SQL

- 在SQL语法中，只要连接列满足连接条件表达式即可执行连接操作，在实际应用中，连接列通常具有可比性，需要满足一定的语义条件。当两个表上存在主码与外码参照关系时，通常执行两个表的主码和外码上的等值连接条件。
- SQL命令示例：

【例4-29】执行NATION表和REGION表上的等值连接操作。

select * from NATION, REGION where N_REGIONKEY=R_REGIONKEY;

– SQL命令解析：NATION表的N_REGIONKEY属性为外码，参照REGION表上的主码R_REGIONKEY，连接条件设置为主、外码相等表示将两个表中REGIONKEY相同的元组连接起来作为查询结果。

select * from NATION INNER JOIN REGION on N_REGIONKEY=R_REGIONKEY;

– --SQL命令解析：等值连接操作还可以采用内连接的语法结构表示。内连接语法如下所示：

<table_name1> INNER JOIN <table_name2>

ON [<table_name1>.]<column_name1> = [<table_name2>.]<column_name2>

- 在SQL命令的WHERE子句中，连接条件可以和其他选择条件组成复合条件，对连接表进行筛选后连接。

第3节 数据查询SQL

- SQL命令示例：

【例4-30】执行表CUSTOMER、ORDERS、LINEITEM上的查询操作。

```
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;
```

– SQL命令解析：如图4-27所示，CUSTOMER、ORDERS、LINEITEM表间存在主码-外码参照关系，CUSTOMER与ORDERS表之间的主-外码等值连接表达式为C_CUSTKEY = O_CUSTKEY，ORDERS表与LINEITEM表之间的主-外码等值连接表达式为L_ORDERKEY = O_ORDERKEY，与其他不同表上的选择条件构成复合条件，完成连接表上的分组聚集计算。

2.自身连接

- 表与自己进行的连接操作称为表的自身连接，简称自连接（self join）。使用自连接可以将自身表的一个镜像当作另一个表来对待，通常采用为表取两个别名的方式实现自连接。
- SQL命令示例：

【例4-31】输出LINEITEM表上订单中L_SHIPINSTRUCT既包含DELIVER IN PERSON又包含TAKE BACK RETURN的订单号。

```
select distinct L1.L_ORDERKEY
from LINEITEM L1, LINEITEM L2
where L1.L_SHIPINSTRUCT='DELIVER IN PERSON'
and L2.L_SHIPINSTRUCT='TAKE BACK RETURN'
and L1.L_ORDERKEY=L2.L_ORDERKEY;
```

  - SQL命令解析：LINEITEM表中一个订单包含多个订单项，每个订单项包含特定的L_SHIPINSTRUCT值，存在一个订单不同的订单项L_SHIPINSTRUCT值既包含DELIVER IN PERSON又包含TAKE BACK RETURN的元组。查询在LINEITEM表中选择L_SHIPINSTRUCT值为DELIVER IN PERSON的元组，再从相同的LINEITEM表以别名的方式选择L_SHIPINSTRUCT值为TAKE BACK RETURN的元组，并且满足两个元组集上L_ORDERKE等值条件。自身连接通过别名将一个表用作多个表，然后按查询需求进行连接。

3.外连接

- 在通常的连接操作中，两个表中满足连接条件的记录才能作为连接结果记录输出。当需要不仅输出连接记录，还要输出不满足连接条件的记录时，可以通过外连接将不满足连接条件的记录对应的连接属性值置为NULL，表示表间记录完整的连接信息。
- 左外连接列出左边关系的所有元组，在右边关系没有满足连接条件的记录时右边关系属性设置空值；右外连接列出右边关系的所有元组，在左边关系中没有满足连接条件的记录时左边关系属性设置为空值；全外连接为左外连接与右外连接的组合。
- SQL命令示例：

【例4-32】输出ORDERS表与CUSTOMER表左外连接与右外连接的结果。

```
select O_ORDERKEY, O_CUSTKEY, C_CUSTKEY
from ORDERS left outer join CUSTOMER on O_CUSTKEY=C_CUSTKEY;
select O_ORDERKEY, O_CUSTKEY, C_CUSTKEY
from ORDERS right outer join CUSTOMER on O_CUSTKEY=C_CUSTKEY;
```

第3节 数据查询SQL

– SQL命令解析：ORDERS表外码O_CUSTKEY参照CUSTOMER表的主码C_CUSTKEY，在执行ORDERS表与CUSTOMER表左外连接操作时，ORDERS表每一个元组都能够从CUSTOMER表中找到所参照主码与外码相等的记录，连接结果集元组数量与ORDERS表行数相同；在执行右外连接时，CUSTOMER表每一个元组与ORDERS表中的元组执行主码与外码属性相等的连接操作，CUSTOMER表元组的C_CUSTKEY属性值在ORDERS表中没有匹配的元组时，ORDERS属性输出为空值。左外连接可以找到在CUSTOMER表中存在，但没有购物记录的用户，其特征是左外连接结果集中CUSTOMER表属性非空而ORDERS表属性为空。左外连接与右外连接结果如图4-24所示。

	O_ORDERKEY	O_CUSTKEY	C_CUSTKEY
1499992	5881445	95725	95725
1499993	5881472	61198	61198
1499994	5881474	33505	33505
1499995	5881476	99238	99238
1499996	5881478	42547	42547
1499997	5881507	134936	134936
1499998	5881537	96011	96011
1499999	5881568	31417	31417
1500000	5881572	61465	61465

	O_ORDERKEY	O_CUSTKEY	C_CUSTKEY
1499997	4	136777	136777
1499998	3	123314	123314
1499999	2	78002	78002
1500000	1	36901	36901
1500001	NULL	NULL	15675
1500002	NULL	NULL	32655
1500003	NULL	NULL	49635
1500004	NULL	NULL	66615
1500005	NULL	NULL	48330

识别僵尸客户

图4-24 ORDERS表与CUSTOMER表左外连接与右外连接结果

- 全外连接命令为full outer join，在本例中，全连接执行结果与左连接相同。

4.多表连接

- 连接操作可以是两表连接，也可以是多表连接。一个位于中心的表与多个表之间的多表连接称为星形连接，对应星形模式。多表连接是数据库的重要技术，表连接顺序对于查询执行性能有重要的影响，也是查询优化技术的重要研究内容。
- SQL命令示例：

  【例4 33】在TPC-H数据库中执行PARTSUPP表与PART表、SUPPLIER表的星形连接操作。

```
select P_NAME,P_BRAND, S_NAME,S_NAME,PS_AVAILQTY
from PART,SUPPLIER,PARTSUPP
where PS_PARTKEY=P_PARTKEY and PS_SUPPKEY=S_SUPPKEY;
```

  – SQL命令解析：PARTSUPP表与PART表、SUPPLIER表存在主码-外码参照关系，PARTSUPP表分别与PART表、part表、SUPPLIER表通过主、外码进行等值连接。SQL命令中FROM子句包含3个连接表名，WHERE子句中包含PARTSUPP表与2个表基于主、外码的等值连接条件，分别对应3个表间连接关系。

第3节 数据查询SQL

- 若使用INNER JOIN语法，则SQL命令如下所示：

```
select P_NAME,P_BRAND, S_NAME,S_NAME,PS_AVAILQTY
from PARTSUPP inner join PART on PS_PARTKEY=P_PARTKEY
inner join SUPPLIER on PS_SUPPKEY=S_SUPPKEY;
```

第3节 数据查询SQL

【例4 34】在TPC-H数据库中执行雪花型连接操作。

```
select C_NAME,O_ORDERDATE,S_NAME,P_NAME,N_NAME,R_NAME,
L_EXTENDEDPRICE*(1-L_DISCOUNT)- PS_SUPPLYCOST * L_QUANTITY as amount
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;
```

创建多表数据统一视图

第3节 数据查询SQL

– SQL命令解析：如图4-25所示，TPC-H数据库是一种典型的雪花型模式，模式以LINEITEM表为中心，通过主码-外码参照关系与其他表连接，而ORDERS、PART、SUPPLIER等表又有下级的参照表，整体上形成雪花形分枝结构。执行雪花型连接时，可以根据数据库模式图，将表间主码-外码参照关系一一转换为表间主码-外码属性间的等值连接表达式，完成雪花型连接操作。

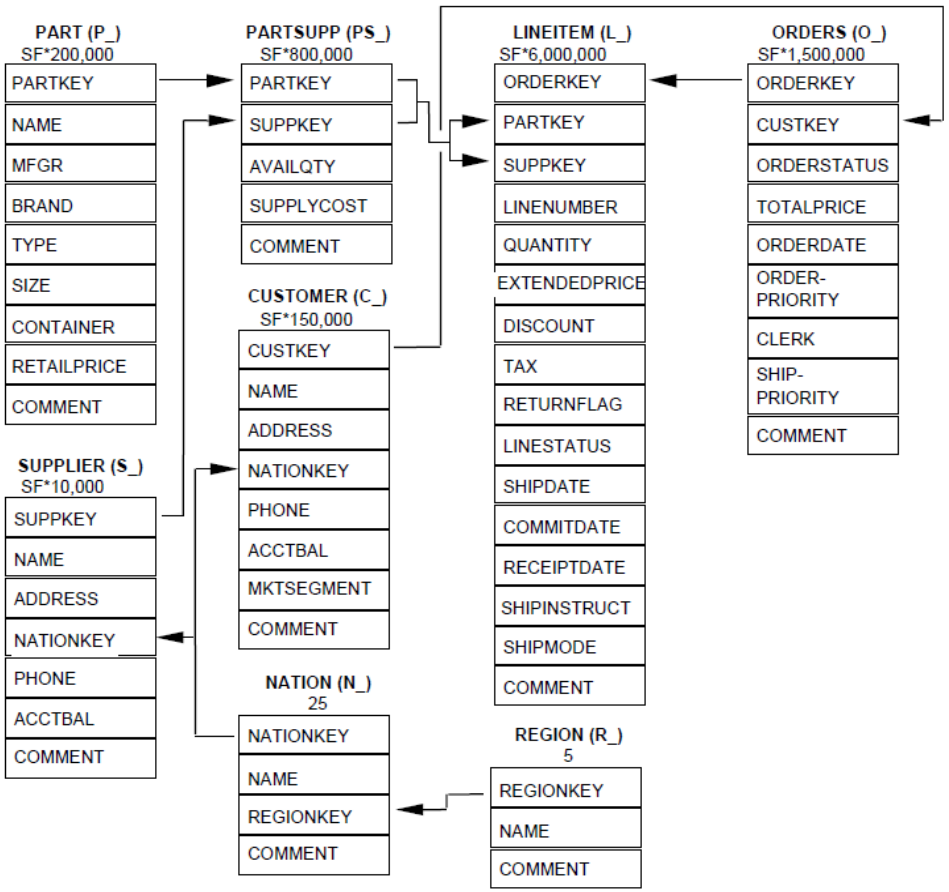
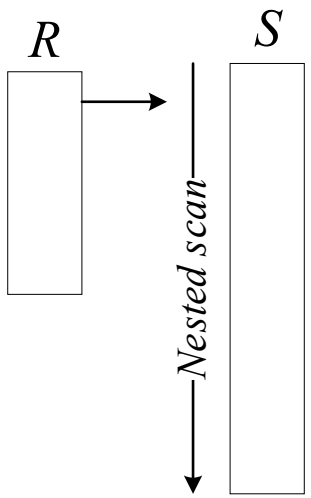


图4-25 TPC-H数据库模式

5.连接操作的基本实现技术

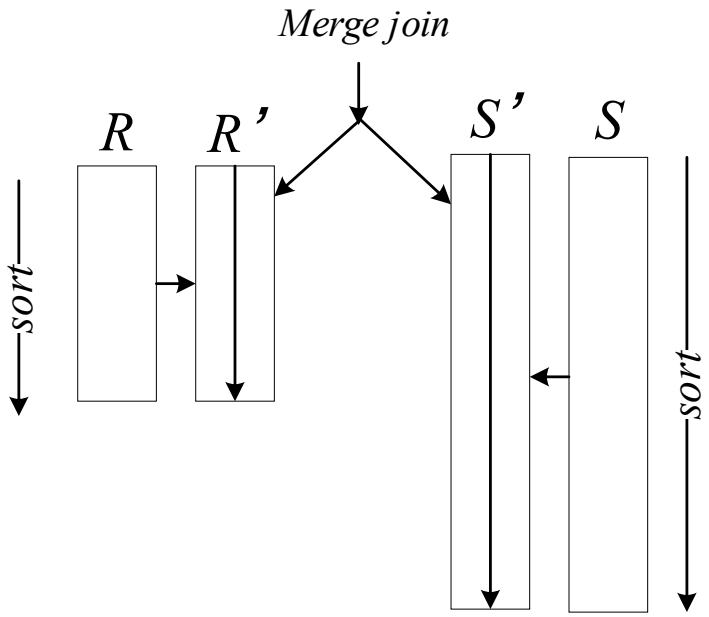
连接是关系数据库中重要的操作，也是数据库中执行代价较高的操作，是关系数据库查询优化的核心技术。连接操作的主要实现技术包括：

- **嵌套循环连接（nested-loop join）**
  - 嵌套循环连接是最基本的连接算法，适用于等值与非等值连接。
  - 如4-26（A）所示，嵌套循环连接将两个连接表用作外表与内表，外表循环扫描每一条记录，内表根据外表当前记录的连接属性在内表中循环扫描每一条记录，找到与外表相匹配的记录连接输出。
  - R和S的记录数记作|R|和|S|，则嵌套循环连接需要执行|R|×|S|次循环比较操作。
  - 在嵌套循环连接中，内表通常为排序表或在连接属性上建有索引，以加速内表记录查找性能。也可以将较小的R表作为内表，R和S通过连接属性上的分区（partition）操作将R表划分为小于cache大小的子表，然后S分区子表与R分区子表执行嵌套循环连接操作，R分区子表采用简单的顺序扫描方法，通过cache内的高性能扫描提高内表记录查找性能。
  - 在新型处理器平台下，嵌套循环连接算法还可以通过SIMD单指令多数据技术一次执行多个内表记录的连接属性比较操作，也可以通过GPU、Phi等众核计算架构加速嵌套循环连接算法性能。



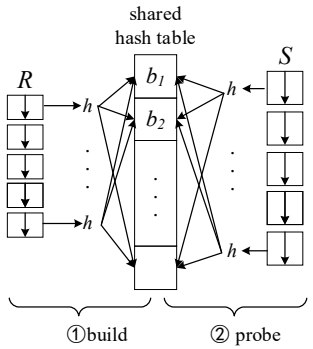
（A）嵌套循环连接

- **排序归并连接（sort-merge join）**
  - 排序归并连接将连接表R和S按连接属性排序，然后从两个关系中各取第一行判断是否符合连接条件，如果符合则输出连接结果，否则比较较小值关系的下一条记录。
  - 如图4-26（B）所示，排序归并连接算法的主要代价在于排序操作，当前代表性的优化技术是基于SIMD优化排序算法，以及使用GPU加速排序操作等。

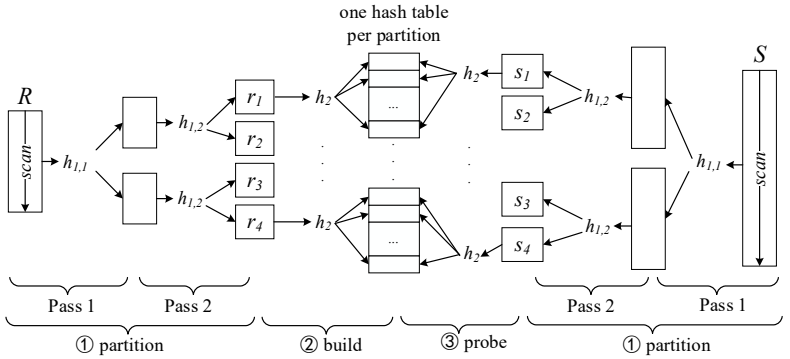


（B）排序归并连接

- **哈希连接（hash join）**
  - 哈希连接算法主要通过哈希匹配查找连接记录。当前代表性的哈希连接算法主要包括基于共享哈希表的无分区哈希连接算法（如图4-26（C）所示）和基于Radix分区的哈希连接算法（如图4-26（D）所示）。
  - 无分区哈希连接算法分为生成（build）和探测（probe）两个阶段，在生成阶段通过扫描R表，为满足选择条件的记录生成哈希表；在探测阶段扫描S表，将S表中的每一条记录按连接属性值在哈希表中进行探测，找到匹配的连接属性值则输出连接记录。在多核处理器平台，哈希表生成阶段可以并行完成。将R表逻辑划分为n个分区，n个线程并行扫描n个分区，并创建线程间共享的哈希表。当线程数量较多时，哈希表由于大量的并发访问冲突而导致哈希表生成性能降低。在探测阶段，n个线程对应的n个分区可以并行执行哈希探测操作，当共享哈希表小于cache时，哈希探测性能较高，当共享哈希表较大时，哈希探测产生较多的cache miss，增加了内存访问代价，降低了哈希探测性能。
  - Radix分区的哈希连接算法对R表和S表的连接属性按相同的Radix进行多趟分区，保证每趟分区时分区数量不超过处理器的TLB大小，优化TLB访问性能，通过多趟Radix分区将R表划分为适合cache大小的分区，然后在分区上创建哈希表。在哈希探测阶段，每个线程执行一对R表和S表分区上的连接操作，通过cache内的哈希表探测优化连接性能。



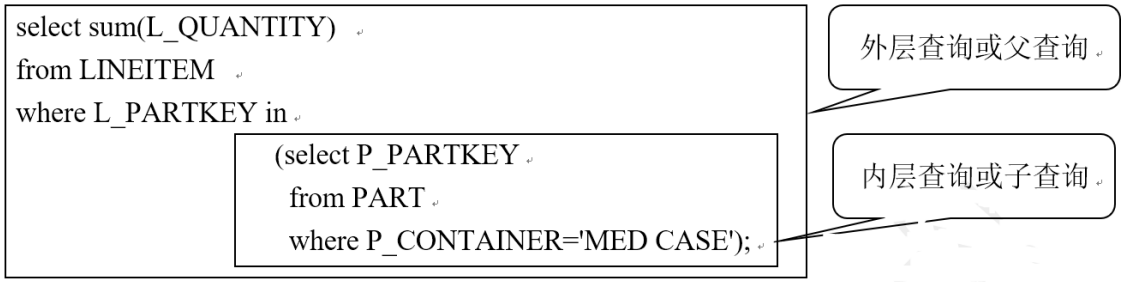
（C）哈希连接



（D）Radix哈希连接

三、嵌套查询

- 在SQL语言中，一个SELECT-FROM-WHERE语句称为一个查询块。当一个查询块嵌套在另一个查询块的WHERE子句时构成了查询嵌套结构，称这种查询为嵌套查询（nested query）。例如：



- 在上面的示例中，子查询（或称为内层查询）select P_PARTKEY from PART where P_CONTAINER='MED CASE'嵌套在父查询（或称为外层查询）中，子查询的结果相当于父查询中IN表达式的集合。值得注意的是，子查询不能直接使用order by子句，order by子句对最终查询结果排序。但本例中当子查询需要输出前10个子查询记录时，子查询中top与order by可以共同使用，即select top 10 P_PARTKEY from PART where P_CONTAINER = 'MED CASE' order by P_PARTKEY。

第3节 数据查询SQL

嵌套查询通过简单查询构造复杂查询，增加SQL的查询能力，降低用户进行复杂数据处理时的难度。从使用特点来看，嵌套查询主要包括以下几类。

1.包含IN谓词的子查询

- 当子查询的结果是一个集合时，通过IN谓词实现父查询WHERE子句中向子查询集合的谓词嵌套判断。
- SQL命令示例：

【例4-35】带有IN子查询的嵌套查询执行。

```
select P_BRAND, P_TYPE, P_SIZE, count (distinct ps_suppkey) as supplier_cnt
from PARTSUPP, PART
where P_PARTKEY = PS_PARTKEY and P_BRAND <> 'Brand#45'
and P_TYPE not like 'MEDIUM POLISHED%'
and P_SIZE in (49, 14, 23, 45, 19, 3, 36, 9)
and PS_SUPPKEY not in (
select S_SUPPKEY
from SUPPLIER
where S_COMMENT like '%Customer%Complaints%'
)
group by P_BRAND, P_TYPE, P_SIZE
order by supplier_cnt desc, P_BRAND, P_TYPE, P_SIZE;
```

第3节 数据查询SQL

- --SQL命令解析：首先执行子查询select S_SUPPKEY from SUPPLIER where S_COMMENT like '%Customer%Complaints%'，得到满足条件的S_SUPPKEY结果集；然后执行外层查询，将子查询结果集作为not in的操作集，排除父查询表PARTSUPP中与内存查询S_SUPPKEY结果集相等的记录，完成父查询。

第3节 数据查询SQL

- 当子查询的查询条件不依赖于父查询时，子查询可以独立执行，这类子查询称为不相关子查询。一种查询执行方法是先执行独立的子查询，然后父查询在子查询的结果集上执行；另一种查询执行方法是将IN谓词操作转换为连接操作，IN谓词执行的列作为连接列，上面的查询可以改写为：

```
select P_BRAND, P_TYPE, P_SIZE, count (distinct PS_SUPPKEY) as
supplier_cnt
from PARTSUPP, PART, SUPPLIER
where P_PARTKEY = PS_PARTKEY and S_SUPPKEY=PS_SUPPKEY
and P_BRAND <> 'Brand#45' and P_TYPE not like 'MEDIUM POLISHED%'
and P_SIZE in (49, 14, 23, 45, 19, 3, 36, 9)
and S_COMMENT not like '%Customer%Complaints%'
group by P_BRAND, P_TYPE, P_SIZE
order by supplier_cnt desc, P_BRAND, P_TYPE, P_SIZE;
```

- --SQL命令解析：嵌套查询条件是not in，改写为连接操作时需要将子查询的条件取反，即，将原子查询中S_COMMENT like '%Customer%Complaints%'改写为S_COMMENT not like '%Customer%Complaints%'，以获得与原始嵌套查询相同的执行结果。

第3节 数据查询SQL

- 【例4-36】通过IN子查询完成CUSTOMER、NATION与REGION表间的查询，统计ASIA地区顾客的数量。

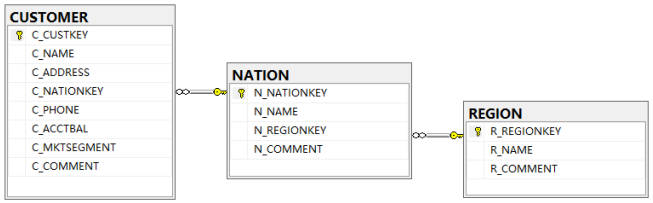


图4-27 CUSTOMER、NATION与REGION表之间的参照关系

```
SELECT count(*) FROM CUSTOMER WHERE C_NATIONKEY IN
    (SELECT N_NATIONKEY FROM NATION WHERE N_REGIONKEY IN
        (SELECT R_REGIONKEY FROM REGION
WHERE R_NAME='ASIA'));
```

- 三个表之间的连接关系为 $customer \xrightarrow{c\_nationkey} nation \xrightarrow{n\_regionkey} region$，查询的谓词条件为最远端表REGION表上的R_NAME='ASIA'，需要将谓词结果投影到连接列R_REGIONKEY上，生成集合（2）传递给NATION表，在NATION表上生成谓词条件N_REGIONKEY IN (2)，投影在连接列N_NATIONKEY上，生成连接列结果集（8,9,12,18,21）传递给CUSTOMER表，最后转换为CUSTOMER表上的谓词条件C_NATIONKEY IN （8,9,12,18,21），完成在父查询中的处理。

第3节 数据查询SQL

- 当前嵌套子查询可以转换为连接查询：

```
select count(*)
from CUSTOMER, NATION, REGION
where C_NATIONKEY=N_NATIONKEY and N_REGIONKEY=R_REGIONKEY
and R_NAME='ASIA';
```

  - SQL命令解析：IN嵌套子查询实现连接列结果集的逐级向上传递，通过对连接列的逐级过滤完成查询处理。上面的连接查询中深色底纹部分实现三个表之间的连接操作，然后可以将连接表上的谓词操作看作连接后的单表上的谓词操作。
- 当子查询的查询条件依赖父查询时，子查询需要迭代地从父查询获得数据才能完成子查询上的处理，这类子查询称为相关子查询，整个查询称为相关嵌套查询。

2.带有比较运算符的相关子查询

- 在相关子查询中，外层父查询提供内层子查询执行时的谓词变量，由外层父查询驱动内存子查询的执行。带有比较运算符的子查询是父查询与子查询之间用比较运算符进行连接，当内存子查询返回结果是单个值时，用>、<、=、>=、<=、!=或<>等比较运算符。

- 【例4-37】带有=比较运算符的子查询。

```
select S_ACCTBAL, S_NAME, N_NAME, P_PARTKEY, P_MFGR, S_ADDRESS,
S_PHONE, S_COMMENT
from PART, SUPPLIER, PARTSUPP, NATION, REGION
where P_PARTKEY = PS_PARTKEY and S_SUPPKEY = PS_SUPPKEY
and P_SIZE = 15 and P_TYPE like '%BRASS'
and S_NATIONKEY = N_NATIONKEY and N_REGIONKEY = R_REGIONKEY
and R_NAME = 'EUROPE'
and PS_SUPPLYCOST = (
select min(PS_SUPPLYCOST)
from PARTSUPP, SUPPLIER, NATION, REGION
where P_PARTKEY = PS_PARTKEY and S_SUPPKEY = PS_SUPPKEY
and S_NATIONKEY = N_NATIONKEY and N_REGIONKEY = R_REGIONKEY
and R_NAME = 'EUROPE'
)
order by S_ACCTBAL desc, N_NAME, S_NAME, P_PARTKEY;
```

第3节 数据查询SQL

- SQL解析：外层查询“PS_SUPPLYCOST =”表达式为子查询结果，即内层子查询输入的min(PS_SUPPLYCOST)结果。内层子查询因缺失P_PARTKEY信息而不能独立执行，该查询是相关子查询。查询执行时，外层查询产生的结果集，下推到内层查询各P_PARTKEY值，下层查询根据外层查询推送P_PARTKEY计算出的结果集，外层查询通过PS_SUPPLYCOST = (…)表达式筛选内层查询的结果，最终产生输出记录。
- 嵌套相关子查询是由外层查询通过数据驱动内层查询执行，每一条外层查询产生的记录调用一次内层查询执行。这种执行方式可以转换为两个独立查询执行结果集的连接操作。

第3节 数据查询SQL

- 改写后的查询如下：

```
WITH ps_supplycostTable (min_supplycost, partkey)
AS
(
    select min(PS_SUPPLYCOST) as min_ps_supplycost, P_PARTKEY
    from PARTSUPP, SUPPLIER, NATION, REGION, PART
    where P_PARTKEY = PS_PARTKEY and S_SUPPKEY = PS_SUPPKEY
and P_SIZE = 15 and P_TYPE like '%BRASS'
and S_NATIONKEY = N_NATIONKEY and N_REGIONKEY = R_REGIONKEY
and R_NAME = 'EUROPE'
    group by P_PARTKEY
)
select
S_ACCTBAL, S_NAME, N_NAME, P_PARTKEY, PS_SUPPLYCOST, P_MFGR,
S_ADDRESS, S_PHONE, S_COMMENT
from PART, SUPPLIER, PARTSUPP, NATION, REGION, ps_supplycostTable
where P_PARTKEY = PS_PARTKEY and S_SUPPKEY = PS_SUPPKEY
and PARTKEY=P_PARTKEY   --增加与派生表partkey连接表达式
and PS_SUPPLYCOST =min_supplycost   --增加与派生表supplycost等值表达式
and P_SIZE = 15 and P_TYPE like '%BRASS'
and S_NATIONKEY = N_NATIONKEY and N_REGIONKEY = R_REGIONKEY
and R_NAME = 'EUROPE'
order by S_ACCTBAL DESC, N_NAME, S_NAME, P_PARTKEY;
```

  - SQL解析：首先将外层查询PART表上的谓词条件下推到内层查询，通过WITH派生表查询所有候选的P_PARTKEY与相应的聚集结果；然后外层查询与WITH派生表连接将比较运算符子查询改写为与WITH派生表属性的等值表达式。

3.带有ANY或ALL谓词的子查询

- 当子查询返回的结果是多值的，比较运算符包含两种语义：与多值的全部结果（ALL）比较，或与多值的某个结果（ANY）比较。使用ANY或ALL谓词时必须同时使用比较运算符，其语义如下：
  - >/>=/</<=/!=/= ANY：大于/大于等于/小于/小于等于/不等于/等于 结果中某个值
  - >/>=/</<=/!=/= ALL：大于/大于等于/小于/小于等于/不等于/等于 结果中所有值
- SQL命令示例：
- 【例4-38】统计LINEITEM表中L_EXTENDEDPRICE大于任何一个中国顾客订单L_EXTENDEDPRICE记录的数量。

```
select count(*) from LINEITEM where L_EXTENDEDPRICE>ANY(
select L_EXTENDEDPRICE from LINEITEM, SUPPLIER, NATION
where L_SUPPKEY=S_SUPPKEY and S_NATIONKEY=N_NATIONKEY
and N_NAME='CHINA');
```

  - SQL解析：子查询选出满足条件的L_EXTENDEDPRICE子集，父查询判断是否当前记录的L_EXTENDEDPRICE大于任意L_EXTENDEDPRICE子集元素。

- >ANY子查询可以改写为子查询中的最小值，即：

```
select count(*) from LINEITEM where L_EXTENDEDPRICE>(
select min(L_EXTENDEDPRICE) from LINEITEM, SUPPLIER, NATION
where L_SUPPKEY=S_SUPPKEY and S_NATIONKEY=N_NATIONKEY
and N_NAME='CHINA');
```

  - SQL解析：大于集合中任一元素值等价于大于集合中最小值。查询改写前需要通过嵌套循环连接算法扫描外层查询的每一条记录，然后将与内层查询的结果集进行比较，查询执行代价较高。改写后的查询先执行内层查询，计算出min聚集结果，然后外层查询与固定的内层min聚集结果比较，查询代价较小。

- 同理，ANY或ALL子查询转换聚集函数的对应关系还包括：

  =ANY等价于IN谓词

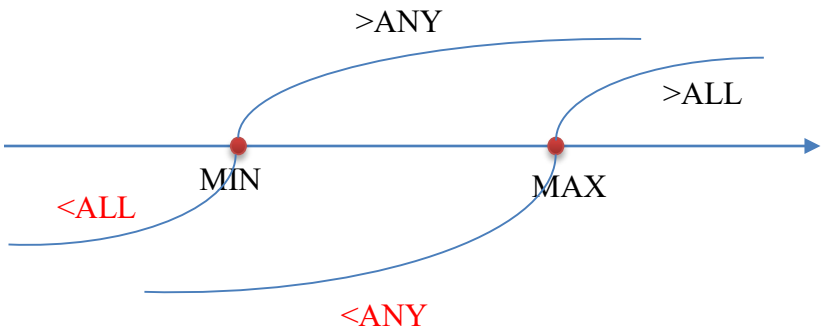
  <>ALL等价于NOT IN谓词

  <(<=)ANY等价于<(<=)MAX谓词

  >(>=)ANY等价于>(>=)MIN谓词

  <(<=)ALL等价于<(<=)MIN谓词

  >(>=)ALL等价于>(>=)MAX谓词



4.带有EXIST谓词的子查询

- 带有EXIST谓词的子查询不返回任何数据，只产生逻辑结果TRUE或FALSE。
- SQL命令示例：
- 【例4-39】分析下面查询exists子查询的作用。

```
select O_ORDERPRIORITY, count(*) as order_count
from ORDERS
where O_ORDERDATE >= '1993-07-01'
and O_ORDERDATE < DATEADD(MONTH, 3,'1993-07-01' )
and exists (
select *
from LINEITEM
where L_ORDERKEY = O_ORDERKEY and L_COMMITDATE < L_RECEIPTDATE )
group by O_ORDERPRIORITY
order by O_ORDERPRIORITY;
```

  - SQL解析：查询在ORDERS表的执行谓词条件，满足谓词条件的ORDERS记录的O_ORDERKEY下推到内层子查询，与LINEITEM表谓词L_COMMITDATE < L_RECEIPTDATE执行结果集进行连接，判断LINEITEM表中是否存在至少存在一个RECEIPTDATE晚于COMMITDATE日期的情况。子查询存在量词exists在内层查询结果为空时，外层where子句返回真值，反之返回假值。如果返回值为真，外层查询执行分组计数操作。

第3节 数据查询SQL

- 该查询可以改写为使用连接运算的SQL语句：

```
select O_ORDERPRIORITY, count (distinct L_ORDERKEY) as order_count
from ORDERS, LINEITEM
where O_ORDERKEY=L_ORDERKEY and O_ORDERDATE >= '1993-07-01'
and O_ORDERDATE < DATEADD (MONTH, 3,'1993-07-01')
and L_COMMITDATE < L_RECEIPTDATE
group by O_ORDERPRIORITY
order by O_ORDERPRIORITY;
```

  - SQL解析：将查询改写为两表连接操作。原始查询判断ORDERS表上是否存在满足连接条件的记录再进行分组计数操作，改写后的查询直接在满足连接条件的记录上执行分组计数操作。需要注意的是，ORDERS表上O_ORDERKEY为主键，而L_ORDERKEY为外键，将查询改写为两表连接方式时一条满足条件的ORDERS表记录会与多个LINEITEM表记录连接，因此需要对连接结果集中的O_ORDERKEY去重统计。

第3节 数据查询SQL

- 【例4-40】查询在没有购买任何商品的顾客的数量。

```
select count(C_CUSTKEY)
from CUSTOMER
where not exists(
select * from ORDERS, LINEITEM
where L_ORDERKEY=O_ORDERKEY and O_CUSTKEY=C_CUSTKEY
);
```

  - SQL解析：通过存在量词not exists检查内层子查询中是否有该用户的订单记录，当子查询结果集为空时向外层查询返回真值，确定该顾客为满足查询条件的顾客。

```
select count(*)
from ORDERS right outer join CUSTOMER on O_CUSTKEY=C_CUSTKEY
where O_ORDERKEY is NULL;
```

  - SQL解析：本查询还可以改写为右连接方式。对ORDERS表与CUSTOMER执行右连接操作，没有购买记录的顾客在右连接结果中ORDERS表属性为空值，可以作为顾客没有购买行为的判断条件。

第3节 数据查询SQL

- 【例4-41】查询在1993年7月起的3个月内没有购买任何商品的顾客的数量。

```
select count(C_CUSTKEY)
from CUSTOMER
where not exists(
select * from ORDERS
where O_CUSTKEY=C_CUSTKEY
and O_ORDERDATE between '1993-07-01' and DATEADD(MONTH,3,'1993-07-01')
);
```

--SQL解析：查询在外层查询扫描CUSTOMER表，将C_CUSTKEY下推到内层查询判断是否存在1993年7月起的3个月内在ORDERS表上的订单记录，如果不存在返回真值，该记录为满足查询条件的记录并计数。

```
select count(distinct C_CUSTKEY)
from (select O_CUSTKEY from orders
where O_ORDERDATE between '1993-07-01' and DATEADD(MONTH,3,'1993-07-01'))
order_3Mon(custkey)
right outer join CUSTOMER on custkey=C_CUSTKEY
where custkey is null;
```

--SQL解析：查询还可以改写为基于右连接的查询命令。首先需要从ORDERS表中筛选出1993年7月起 3个月内的订单记录，将这部分查询命令作为派生表嵌入from子句中，并设置派生表的名称和属性名order_3Mon(custkey)，然后将派生表与CUSTOMER表做右连接操作，以派生表custkey为空作为判断C_CUSTKEY在派生表中没有订单的依据。

- 查询的结果还可以通过集合操作来验证。

```
select C_CUSTKEY from CUSTOMER
except
select distinct O_CUSTKEY from ORDERS, CUSTOMER
where O_CUSTKEY=C_CUSTKEY
and O_ORDERDATE between '1993-07-01' and DATEADD(MONTH,3,'1993-07-01');
```

  - SQL解析：首先选择CUSTOMER表中所有C_CUSTKEY集合，然后再选出ORDERS表中1993年7月起 3个月内的订单的去重O_CUSTKEY集合，两个集合的差运算结果即为CUSTOMER表中1993年7月起 3个月内没有产生订单的顾客。

【案例实践 9】

创建TPC-H数据库，导入TPC-H数据，调试并分析嵌套查询Q2、Q4、Q7、Q8、Q9、Q11、Q13、Q15、Q16、Q17、Q18、Q20、Q21、Q22，尝试能否将嵌套查询改写为非嵌套查询命令，并对比不同形式SQL命令执行的效率。

四、集合查询

- 当多个查询的结果集具有相同的列和数据类型时，查询结果集之间可以进行集合的并（UNIOIN）、交（INTERSECT）和差（EXCEPT）操作。参与集合操作的原始表的结构可以不同，但结果集需要具有相同的结构，即相同的列数，对应列的数据类型相同。
- **SQL命令示例： 集合并运算**

【例4-42】查询LINEITEM表中L_SHIPMODE模式为AIR或AIR REG，以及L_SHIPINSTRUCT方式为DELIVER IN PERSON的订单号。

```
select distinct L_ORDERKEY from LINEITEM
where L_SHIPMODE in ('AIR','AIR REG')
union
select distinct L_ORDERKEY from LINEITEM
where L_SHIPINSTRUCT ='DELIVER IN PERSON'

select distinct L_ORDERKEY from LINEITEM
where L_SHIPMODE in ('AIR','AIR REG')
union all
select distinct L_ORDERKEY from LINEITEM
where L_SHIPINSTRUCT ='DELIVER IN PERSON'
```

– SQL解析：union将两个查询的结果集进行合并，union all保留两个结果集中全部的结果，包括重复的结果，union则在结果集中去掉重复的结果。

- 当在相同的表上执行union操作时，可以将union操作转换为选择谓词的或or表达式，如：

```
select distinct L_ORDERKEY from LINEITEM
where L_SHIPMODE in ('AIR','AIR REG') or L_SHIPINSTRUCT
='DELIVER IN PERSON';
```

- **SQL命令示例： 集合交运算**

【例4-43】查询CONTAINER为WRAP BOX、MED CASE、JUMBO PACK，并且PS_AVAILQTY低于1000的产品名称。

```
select P_NAME from PART
where P_CONTAINER in ('WRAP BOX','MED CASE','JUMBO PACK')
intersect
select P_NAME from PART, PARTSUPP
where P_PARTKEY=PS_PARTKEY and PS_AVAILQTY<1000;
```

– SQL解析：首先在PART表上投影出CONTAINER为WRAP BOX、MED CASE、JUMBO PACK的P_NAME；然后连接PART与PARTSUPP表，按PS_AVAILQTY<1000条件筛选出P_NAME，由于PART与PARTSUPP的主码-外码参照关系，P_NAME存在重复的记录；最后执行两个集合的并操作，获得满足两个集合条件的P_NAME结果集，并通过集合操作消除重复的P_NAME。

- 本例可以改写为基于连接操作的查询，但需要注意的是P_PARTKEY在输出时需要通过distinct消除连接操作产生的重复值，改写的SQL命令如下：

```
select distinct P_PARTKEY from PART, PARTSUPP
where P_PARTKEY=PS_PARTKEY and PS_AVAILQTY<1000
and P_CONTAINER in ('WRAP BOX','MED CASE','JUMBO PACK')
order by P_PARTKEY;
```

- **SQL命令示例： 集合差运算**

【例4-44】查询ORDERS表中O_ORDERPRIORITY类型为1-URGENT和2-HIGH，但O_ORDERSTATUS状态不为F的订单号。

```
Select O_ORDERKEY from ORDERS
where O_ORDERPRIORITY in ('1-URGENT','2-HIGH')
except
select O_ORDERKEY from ORDERS
where O_ORDERSTATUS='F';

select O_ORDERKEY from ORDERS
where O_ORDERPRIORITY in ('1-URGENT','2-HIGH') and not O_ORDERSTATUS='F';
```

– SQL解析：ORDERS表的主码为O_ORDERKE，集合差操作的两个集合操作都是面向相同记录的不同属性，可以将差操作改写为第一个谓词与第二个谓词取反的合取表达式查询命令。

- **SQL命令示例： 多值列集合差运算**

【例4-45】查询LINEITEM表中执行L_SHIPMODE模式为AIR或AIR REG，但L_SHIPINSTRUCT方式不是DELIVER IN PERSON的订单号。

- 按查询要求将查询条件L_SHIPMODE模式为AIR或AIR REG作为一个集合，查询条件L_SHIPINSTRUCT方式是DELIVER IN PERSON作为另一个集合，然后求集合差操作。查询命令如下：

```
select L_ORDERKEY from LINEITEM
where L_SHIPMODE in ('AIR','AIR REG')
except
select L_ORDERKEY from LINEITEM
where L_SHIPINSTRUCT ='DELIVER IN PERSON'
```

- 由于执行的是集合运算，集合的结果自动去重。
- 当改写此查询时，在输出的L_ORDERKEY前面需要手工增加distinct语句对结果集去重，差操作集合谓词条件改为L_SHIPINSTRUCT! ='DELIVER IN PERSON'，查询命令如下：

```
select distinct L_ORDERKEY from LINEITEM
where L_SHIPMODE in ('AIR','AIR REG') and L_SHIPINSTRUCT! ='DELIVER IN PERSON';
```

- 重写后的查询与集合差操作查询结果不一致！

第3节 数据查询SQL

- 通过对表中记录的分析可知，如图4-28所示，L_ORDERKEY列为多值列，相同的L_ORDERKEY对应多条记录，集合差运算的语义对应一个订单下的订单项需要满足L_SHIPMODE模式为AIR或AIR REG，但该订单的L_SHIPINSTRUCT方式不能是DELIVER IN PERSON。改写的查询判断的是同一记录不同字段需要满足的条件，与查询语义不符，因此查询结果错误。

	L_ORDERKEY	L_SHIPINSTRUCT	L_SHIPMODE
48756	48546	COLLECT COD	AIR
48757	48546	DELIVER IN PERSON	REG AIR
48758	48546	TAKE BACK RETURN	FOB
48759	48547	COLLECT COD	SHIP
48760	48547	NONE	AIR
48761	48548	NONE	FOB
48762	48548	TAKE BACK RETURN	MAIL
48763	48548	TAKE BACK RETURN	FOB
48764	48548	TAKE BACK RETURN	FOB
48765	48548	DELIVER IN PERSON	TRUCK
48766	48548	NONE	RAIL
48767	48548	NONE	REG AIR

图4-28多键值列上的集合差操作

第3节 数据查询SQL

- 根据集合差运算查询的语义，第一种改写方式是通过IN操作判断满足L_SHIPMODE模式为AIR或AIR REG条件的订单号不能在满足L_SHIPINSTRUCT方式是DELIVER IN PERSON的订单号集合中。查询命令如下：

```
Select distinct L_ORDERKEY from LINEITEM
where L_SHIPMODE in ('AIR','AIR REG') and L_ORDERKEY not in (
select L_ORDERKEY from LINEITEM
where L_SHIPINSTRUCT ='DELIVER IN PERSON');
```

- 第二种改写方法是通过NOT EXISTS语句判断满足L_SHIPMODE模式为AIR或AIR REG条件的当前记录订单号是否存在满足L_SHIPINSTRUCT方式是DELIVER IN PERSON的情况。

```
Select distinct L_ORDERKEY from LINEITEM L1
where L_SHIPMODE in ('AIR','AIR REG') and not exists (
select * from LINEITEM L2 where L1.L_ORDERKEY=L2.L_ORDERKEY
and L_SHIPINSTRUCT ='DELIVER IN PERSON');
```

- 集合差运算的改写较为复杂，要根据表的数据情况具体分析，并进行验证。

五、基于派生表查询

- 当一个复杂的查询需要不同的数据集进行运算时，可以通过派生表和with子句定义查询块或查询子表。
- 当子查询出现在FROM子句中，子查询起到临时派生表的作用，成为主查询的临时表对象。
- SQL命令示例：

【例4-46】分析下面查询中派生表的作用。

```
select C_COUNT, count (*) as custdist
from (
select C_CUSTKEY, count (O_ORDERKEY)
from CUSTOMER left outer join ORDERS on C_CUSTKEY = O_CUSTKEY
and O_COMMENT not like '%special%requests%'
group by C_CUSTKEY
) as C_ORDERS (C_CUSTKEY, C_COUNT)
group by C_COUNT
order by CUSTDIST desc, C_COUNT desc;
```

第3节 数据查询SQL

- --SQL命令解析：from子句中由一个完整的查询定义派生表，命名为C_ORDERS，按C_CUSTKEY分组统计订单号数量。由于派生表输入属性中包含分组聚集结果，因此需要派生表指定表名与列名，然后在派生表上完成按分组订单数量分组聚集操作，实现对分组聚集结果再分组聚集计算。
- 派生表的功能也可以定义公用表表达式来实现。公用表表达式用于指定临时命名的结果集，将子查询定义为公用表表达式，在使用时要求公用表表达式后面紧跟着使用公用表表达式的SQL命令。上例查询命令将派生表查询块用with表达式定义，简化查询命令结构：

```
WITH C_ORDERS (C_CUSTKEY, C_COUNT)
AS (
select C_CUSTKEY, count (O_ORDERKEY)
from CUSTOMER left outer join ORDERS on C_CUSTKEY = O_CUSTKEY
and O_COMMENT not like '%special%requests%'
group by C_CUSTKEY)
select C_COUNT, count (*) as custdist
from C_ORDERS group by C_COUNT
order by CUSTDIST desc, C_COUNT desc;
```

第3节 数据查询SQL

- SQL命令示例：

【例4-47】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。

```
select C_CUSTKEY, count (*) AS counter,
ROW_NUMBER() over (ORDER BY counter) as RowNum
from ORDERS, CUSTOMER
where O_CUSTKEY=C_CUSTKEY
group by C_CUSTKEY
order by counter, C_CUSTKEY;
```

- SQL解析：聚集表达式上的ROW_NUMBER函数无效，无法完成序列操作。

```
WITH custkey_counter(CUSTKEY, counter)
AS (
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;
```

- SQL解析：将带有分组聚集命令的查询块定义为WITH表达式，聚集列定义为表达式属性，然后在查询中通过from子句访问该WITH表达式，通过ROW_NUMBER函数为查询结果分配行号。
- WITH表达式一方面可以将复杂查询中的查询块预先定义，简化查询主体结构，另一方面可以通过表达式实现一些对聚集结果列的处理任务。

Q & A