TABLE  可以保存数据的结构。所有数据以行和列的形式进行组织。每列的数据类型都是显式定义的。

INDEX  用来加速对表进行查找的对象。INDEX的作用与书籍的索引方式类似，将特别选择的相对较少的信息进行复制和排序，以进行快速引用，并将它与表中的位置反向联系起来，以支持快速查找源表中的行。

VIEW  是可以用来对表进行查找和交互的"过滤器"，但它自己并不保存数据，只是作为一个或多个表上的"窗口"。VIEW对象可用来掩盖底层表的逻辑，使用VIEW对象的原因很多：可能为了简化业务逻辑，或者通过隐藏真实信息来源增加一层安全性。VIEW可用来显示表的特定部分，与此同时，隐藏了同一个表的其他部分。

SEQUENCE  它是一个计数器，经常用来生成唯一的数值，当将新的行加入到表中时，可以用作行的标识符。

SYNONYM  数据库中另一个对象的别名，经常用来为表或视图指定一个可选的名称。

CONSTRAINT  由程序员定义的一小段逻辑，用来指示特定的表如何接受、修改或拒绝输入数据。

USERS  数据库对象的"所有者"。

ROLES  可以授权给用户的一种或多种权限。

[**ORACLE基本数据类型总结**](http://www.cnblogs.com/kerrycode/p/3265120.html)

ORACLE基本数据类型（亦叫内置数据类型 built-in datatypes)可以按类型分为：字符串类型、数字类型、日期类型、LOB类型、LONG RAW& RAW类型、ROWID & UROWID类型。

**一 字符串类型**

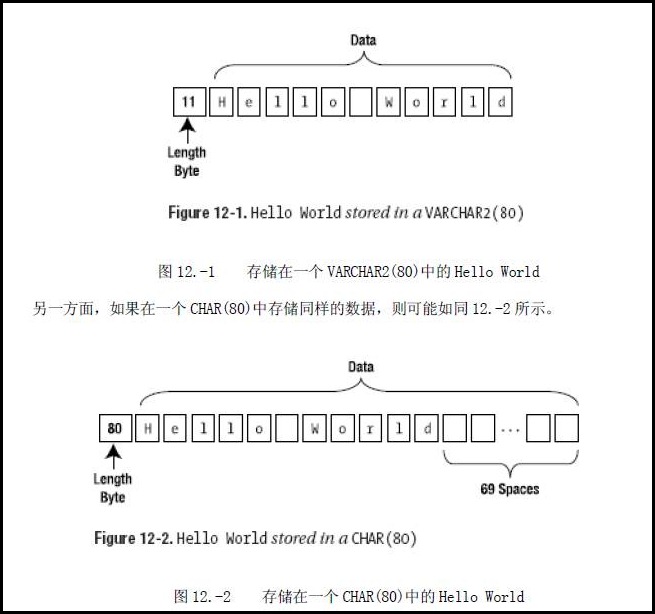
    字符串数据类型还可以依据存储空间分为固定长度类型（CHAR/NCHAR) 和可变长度类型（VARCHAR2/NVARCHAR2)两种.

    所谓固定长度：是指虽然输入的字段值小于该字段的限制长度，但是实际存储数据时，会先自动向右补足空格后，才将字段值的内容存储到数据块中。这种方式虽然比较浪费空间，但是存储效率较可变长度类型要好。同时还能减少数据行迁移情况发生。

所谓可变长度：是指当输入的字段值小于该字段的限制长度时，直接将字段值的内容存储到数据块中，而不会补上空白，这样可以节省数据块空间。

**1.1：CHAR类型 CHAR(size [BYTE | CHAR])**

CHAR类型，定长字符串，会用空格填充来达到其最大长度。非NULL的CHAR（12）总是包含12字节信息。CHAR字段最多可以存储2,000字节的信息。如果创建表时，不指定CHAR长度，则默认为1。另外你可以指定它存储字节或字符，例如 CHAR(12 BYTYE) CHAR(12 CHAR).一般来说默认是存储字节，你可以查看数据库参数 NLS\_LENGTH\_SEMANTICS的值。



**1.2： NCHAR类型**

这是一个包含UNICODE格式数据的定长字符串。NCHAR字段最多可以存储2,000字节的信息。它的最大长度取决于国家字符集。另外查询时，如果字段是NCHAR类型，则需要如下书写

SELECT translated\_description FROM product\_descriptions

WHERE translated\_name = N'LCD Monitor 11/PM';

**1.3 VARCHAR类型**

不要使用VARCHAR数据类型。使用VARCHAR2数据类型。虽然VARCHAR数据类型目前是VARCHAR2的同义词，VARCHAR数据类型将计划被重新定义为一个单独的数据类型用于可变长度的字符串相比，具有不同的比较语义。

**1.4： VARCHAR2类型**

变长字符串，与CHAR类型不同，它不会使用空格填充至最大长度。VARCHAR2最多可以存储4,000字节的信息。

**1.5： NVARCHAR2类型**

这是一个包含UNICODE格式数据的变长字符串。 NVARCHAR2最多可以存储4,000字节的信息。

**二. 数字类型**

**2.1 NUMBER类型**

NUMBER(P,S)是最常见的数字类型，可以存放数据范围为10^130~10^126（不包含此值)，需要1~22字节(BYTE)不等的存储空间。

P 是Precison的英文缩写，即精度缩写，表示有效数字的位数，最多不能超过38个有效数字

S是Scale的英文缩写，可以使用的范围为-84~127。Scale为正数时，表示从小数点到最低有效数字的位数，它为负数时，表示从最大有效数字到小数点的位数

**2.2 INTEGER类型**

INTEGER是NUMBER的子类型，它等同于NUMBER（38,0），用来存储整数。若插入、更新的数值有小数，则会被四舍五入。

**2.3 浮点数**

浮点数可以有一个十进制数点任何地方从第一个到最后一个数字，或者可以在所有有没有小数点。指数可能（可选） 用于以下数量增加的范围 （例如， 1.777e-20)。刻度值不适用于浮点数字，因为可以显示在小数点后的位数的数量不受限制。

二进制浮点数不同数量的值由 Oracle 数据库内部存储的方式。使用小数精度数存储值。完全相同号码存储范围和数量由支持的精度内的所有文本。正是因为使用小数精度（数字 0 到 9） 表示文本存储文本。使用二进制精度 （数字 0 和 1） 存储二进制浮点数。这种存储方案不能代表所有确切地使用小数精度的值。频繁地，将值从十进制转换为二进制的精度时出现的错误时撤消值回从二进制转换为十进制精度。在字面 0.1 是一个这样的例子。

Oracle 数据库提供了专为浮点数的两种数值数据类型：

**BINARY\_FLOAT**

BINARY\_FLOAT 是 32 位、 单精度浮点数字数据类型。可以支持至少6位精度,每个 BINARY\_FLOAT 的值需要 5 个字节，包括长度字节。

**BINARY\_DOUBLE**

BINARY\_DOUBLE 是为 64 位，双精度浮点数字数据类型。每个 BINARY\_DOUBLE 的值需要 9 个字节，包括长度字节。

在数字的列中，浮点数有小数精度。在 BINARY\_FLOAT 或 BINARY\_DOUBLE 的列中，浮点数有二进制的精度。二进制浮点数支持的特殊值无穷大和 NaN （不是数字）。

您可以指定列在表 2-4 范围内的浮点数。"数字文本"中定义了用于指定浮点数的格式。

Table 2-3 Floating Point Number Limits

|  |  |  |
| --- | --- | --- |
| **Value** | **Binary-Float** | **Binary-Double** |
| Maximum positive finite value | 3.40282E+38F | 1.79769313486231E+308 |
| Minimum positive finite value | 1.17549E-38F | 2.22507485850720E-308 |

**2.5 FLOAT类型**

FLOAT类型也是NUMBER的子类型。

Float(n),数 n 指示位的精度，可以存储的值的数目。N 值的范围可以从 1 到 126。若要从二进制转换为十进制的精度，请将 n 乘以 0.30103。要从十进制转换为二进制的精度，请用 3.32193 乘小数精度。126 位二进制精度的最大值是大约相当于 38 位小数精度。

**三.** **日期类型**

日期类型用于存储日期数据，但是并不是使用一般的格式（2012-08-08）直接存储到数据库的。

**3.1 DATE类型**

DATE是最常用的数据类型，日期数据类型存储日期和时间信息。虽然可以用字符或数字类型表示日期和时间信息，但是日期数据类型具有特殊关联的属性。为每个日期值，Oracle 存储以下信息： 世纪、 年、 月、 日期、 小时、 分钟和秒。一般占用7个字节的存储空间。

**3.2 TIMESTAMP类型**

这是一个7字节或12字节的定宽日期/时间数据类型。它与DATE数据类型不同，因为TIMESTAMP可以包含小数秒，带小数秒的TIMESTAMP在小数点右边最多可以保留9位

**3.3 TIMESTAMP WITH TIME ZONE类型**

这是TIMESTAMP类型的变种，它包含了时区偏移量的值

**3.4 TIMESTAMP WITH LOCAL TIME ZONE类型**

**3.5 INTERVAL YEAR TO MOTH**

**3.6 INTERVAL DAY TO SECOND**

**四.** **LOB类型**

内置的LOB数据类型包括BLOB、CLOB、NCLOB、BFILE（外部存储）的大型化和非结构化数据，如文本、图像、视屏、空间数据存储。BLOB、CLOB、NCLOB类型

**4.1 CLOB 数据类型**

   它存储单字节和多字节字符数据。支持固定宽度和可变宽度的字符集。CLOB对象可以存储最多 (4 gigabytes-1) \* (database block size) 大小的字符

**4.2 NCLOB 数据类型**

   它存储UNICODE类型的数据，支持固定宽度和可变宽度的字符集，NCLOB对象可以存储最多(4 gigabytes-1) \* (database block size)大小的文本数据。

**4.3 BLOB 数据类型**

   它存储非结构化的二进制数据大对象，它可以被认为是没有字符集语义的比特流，一般是图像、声音、视频等文件。BLOB对象最多存储(4 gigabytes-1) \* (database block size)的二进制数据。

**4.4 BFILE 数据类型**

二进制文件，存储在数据库外的系统文件，只读的，数据库会将该文件当二进制文件处理. Used to store information on external file.

**五.** **RAW & LONG RAW类型**

**5.1 LONG类型**

 它存储变长字符串，最多达2G的字符数据（2GB是指2千兆字节， 而不是2千兆字符），与VARCHAR2 或CHAR 类型一样，存储在LONG 类型中的文本要进行字符集转换。ORACLE建议开发中使用CLOB替代LONG类型。支持LONG 列只是为了保证向后兼容性。CLOB类型比LONG类型的限制要少得多。 LONG类型的限制如下：

1.一个表中只有一列可以为LONG型。(Why?有些不明白)

2.LONG列不能定义为主键或唯一约束，

3.不能建立索引

4.LONG数据不能指定正则表达式。

5.函数或存储过程不能接受LONG数据类型的参数。

6.LONG列不能出现在WHERE子句或完整性约束（除了可能会出现NULL和NOT NULL约束）

**5.2 LONG RAW 类型**，能存储2GB 的原始二进制数据（不用进行字符集转换的数据）

**5.3 RAW类型**

用于存储二进制或字符类型数据，变长二进制数据类型，这说明采用这种数据类型存储的数据不会发生字符集转换。这种类型最多可以存储2,000字节的信息

**六.** **ROWID & UROWID类型**

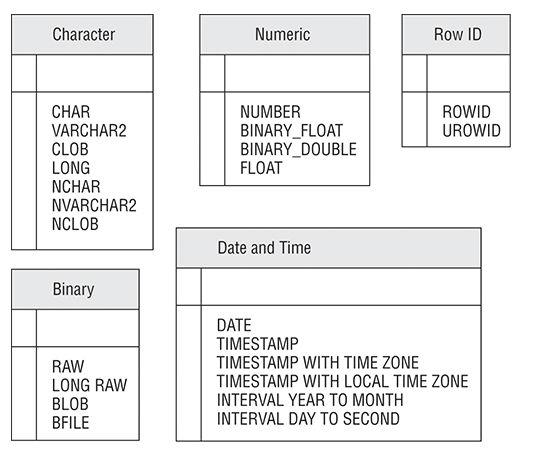
在数据库中的每一行都有一个地址。然而，一些表行的地址不是物理或永久的，或者不是ORACLE数据库生成的。

例如，索引组织表行地址存储在索引的叶子，可以移动。

例如,外部表的ROWID（如通过网关访问DB2表）不是​​标准的ORACLE的rowid。

ORACLE使用通用的ROWID（UROWIDs）的存储地址的索引组织表和外表。索引组织表有逻辑urowids的，和国外表的外urowids，。UROWID这两种类型的存储在ROWID伪（堆组织的表的物理行id）。

创建基于逻辑的rowid在表中的主键。逻辑的rowid不会改变，只要主键不改变。索引组织表的ROWID伪UROWID数据类型。你可以访问这个伪列，你会堆组织表的ROWID伪（即使用一个SELECT ...ROWID语句）。如果你想存储的rowid索引组织表，那么你就可以定义一列的表型UROWID到列检索值的ROWID伪。



约束分为4种，分别是主键（Primary Key），检查（Check），唯一性（Unique），外键（Foreign Key）。另外还有两种是NOT NULL和REF，REF就是其中的一列或者几列是另外一张表中的值。

|  |
| --- |
| **Oracle数据库的完整性约束规则详解** |
|  |
| |  | | --- | | 完整性约束用于增强数据的完整性，Oracle提供了5种完整性约束：  　　　 Check 　　　 NOT NULL 　　　 Unique 　　　 Primary 　　　 Foreign key  完整性约束是一种规则，不占用任何数据库空间。完整性约束存在数据字典中，在执行SQL或PL/SQL期间使用。用户可以指明约束是启用的还是禁用的，当约束启用时，他增强了数据的完整性，否则，则反之，但约束始终存在于数据字典中。禁用约束，使用ALTER语句:  ALTER TABLE table\_name DISABLE CONSTRAINT constraint\_name;  或  ALTER TABLE policies DISABLE CONSTRAINT chk\_gender  如果要重新启用约束：  ALTER TABLE policies ENABLE CONSTRAINT chk\_gender  删除约束:  ALTER TABLE table\_name DROP CONSTRAINT constraint\_name  或  ALTER TABLE policies DROP CONSTRAINT chk\_gender;  Check 约束  在数据列上Check 约束需要一个特殊的布尔条件或者将数据列设置成TRUE，至少一个数据列的值是NULL，Check约束用于增强表中数据内容的简单的商业规则。用户使用 Check约束保证数据规则的一致性。Check约束可以涉及该行同属Check约束的其他数据列但不能涉及其他行或其他表，或调用函数SYSDATE, UID,USER,USERENV。如果用户的商业规则需要这类的数据检查，那么可以使用触发器。Check约束不保护LOB数据类型的数据列和对象、嵌套表、VARRY、ref等。单一数据列可以有多个Check约束保护，一个Check约束可以保护多个数据列。创建表的Check约束使用CREATE TABLE语句，更改表的约束使用ALTER TABLE语句。  语法：  CONSTRAINT [constraint\_name] CHECK (condition);  Check约束可以被创建或增加为一个表约束，当Check约束保护多个数据列时，必须使用表约束语法。约束名是可选的并且如果这个名字不存在，那么oracle将产生一个以SYS\_开始的唯一的名字。例：  CREATE TABLE policies (policy\_id NUMBER, holder\_name VARCHAR2(40), gender VARCHAR2(1) constraint chk\_gender CHECK (gender in ('M','F'), marital\_status VARCHAR2(1), date\_of\_birth DATE, constraint chk\_marital CHECK (marital\_status in('S','M','D','W')) );  NOT NULL约束  NOT NULL约束应用在单一的数据列上，并且他保护的数据列必须要有数据值。缺省状况下，ORACLE允许任何列都可以有NULL值。某些商业规则要求某数据列必须要有值，NOT NULL约束将确保该列的所有数据行都有值。例：  CREATE TABLE policies (policy\_id NUMBER, holder\_name VARCHAR2(40) NOT NULL, gender VARCHAR2(1), marital\_status VARCHAR2(1), date\_of\_birth DATE NOT NULL );  对于NOT NULL的ALTER TABLE语句与其他约束稍微有点不同。  ALTER TABLE policies MODIFY holder\_name NOT NULL  唯一性约束(Unique constraint)  唯一性约束可以保护表中多个数据列，保证在保护的数据列中任何两行的数据都不相同。唯一性约束与表一起创建，在唯一性约束创建后，可以使用ALTER TABLE语句修改。语法：  column\_name data\_type CONSTRAINT constraint\_name UNIQUE  如果唯一性约束保护多个数据列，那么唯一性约束要作为表约束增加。语法如下：  CONSTRAINT constraint\_name (column) UNIQUE USING INDEX TABLESPACE (tablespace\_name) STORAGE (stored clause)  唯一性约束由一个B-tree索引增强，所以可以在USING子串中为索引使用特殊特征，比如表空间或存储参数。CREATE TABLE语句在创建唯一性约束的同时也给目标数据列建立了一个唯一的索引。  CREATE TABLE insured\_autos (policy\_id NUMBER CONSTRAINT pk\_policies PRIMARY KEY, vin VARCHAR2(10), coverage\_begin DATE, coverage\_term NUMBER, CONSTRAIN unique\_auto UNIQUE (policy\_id,vin) USING INDEX TABLESPACE index STORAGE (INITIAL 1M NEXT 10M PCTINCREASE 0) );  用户可以禁用未以性约束，但他仍然存在，禁用唯一性约束使用ALTER TABLE 语句。  ALTER TABLE insured\_autos DISABLE CONSTRAIN unique\_name;  删除唯一性约束，使用ALTER TABLE....DROP CONSTRAIN语句：  ALTER TABLE insured\_autos DROP CONSTRAIN unique\_name;  注意用户不能删除在有外部键指向的表的唯一性约束。这种情况下用户必须首先禁用或删除外部键（foreign key)。删除或禁用唯一性约束通常同时删除相关联的唯一索引，因而降低了数据库性能。经常删除或禁用唯一性约束有可能导致丢失索引带来的性能错误。要避免这样错误，可以采取下面的步骤：  1、在唯一性约束保护的数据列上创建非唯一性索引。  2、添加唯一性约束。  主键(Primary Key)约束  表有唯一的主键约束。表的主键可以保护一个或多个列，主键约束可与NOT NULL约束共同作用于每一数据列。NOT NULL约束和唯一性约束的组合将保证主键唯一地标识每一行。像唯一性约束一样，主键由B-tree索引增强。创建主键约束使用CREATE TABLE语句与表一起创建，如果表已经创建了，可以使用ALTER TABLE语句。  CREATE TABLE policies (policy\_id NUMBER CONSTRAINT pk\_policies PRIMARY KEY, holder\_name VARCHAR2(40), gender VARCHAR2(1), marital\_status VARCHAR2(1), date\_of\_birth DATE );  与唯一性约束一样，如果主键约束保护多个数据列，那么必须作为一个表约束创建。  CREATE TABLE insured\_autos (policy\_id NUMBER, vin VARCHAR2(40), coverage\_begin DATE, coverage\_term NUMBER, CONSTRAINT pk\_insured\_autos PRIMARY KEY (policy\_id,vin) USING INDEX TABLESPACE index STORAGE (INITIAL 1M NEXT 10M PCTINCREASE 0) );  禁用或删除主键必须与ALTER TABLE 语句一起使用:  ALTER TABLE policies DROP PRIMARY KEY;  或  ALTER TABLE policies DISABLE PRIMARY KEY;  外部键约束（Foreign key constraint)  外部键约束保护一个或多个数据列，保证每个数据行的数据包含一个或多个null值，或者在保护的数据列上同时拥有主键约束或唯一性约束。引用（主键或唯一性约束）约束可以保护同一个表，也可以保护不同的表。与主键和唯一性约束不同外部键不会隐式建立一个B-tree索引。在处理外部键时，我们常常使用术语父表（parent table)和子表（child table),父表表示被引用主键或唯一性约束的表，子表表示引用主键和唯一性约束的表。创建外部键使用CREATE TABLE语句，如果表已经建立了，那么使用ALTER TABLE语句。  CREATE TABLE insured\_autos (policy\_id NUMBER CONSTRAINT policy\_fk REFERENCE policies(policy\_id ON DELETE CASCADE, vin VARCHAR2(40), coverage\_begin DATE, coverage\_term NUMBER, make VARCHAR2(30), model VARCHAR(30), year NUMBER, CONSTRAIN auto\_fk FROEIGN KEY (make,model,year) REFERENCES automobiles (make,model,year) ON DELETE SET NULL );  ON DELETE子串告诉ORACLE如果父纪录（parent record)被删除后，子记录做什么。缺省情况下禁止在子记录还存在的情况下删除父纪录。  外部键和NULL值  在外部键约束保护的数据列中NULL值的处理可能产生不可预料的结果。ORACLE 使用ISO standar Match None规则增强外部键约束。这个规则规定如果任何外部键作用的数据列包含有一个NULL值，那么任何保留该键的数据列在父表中没有匹配值。  比如，在父表AUTOMOBILES中，主键作用于数据列MAKE，MODEL，YEAR上，用户使用的表INSURED\_AUTOS有一个外部约束指向AOTOMOBILES，注意在INSURES\_AUTOS中有一数据行的MODEL列为NULL值，这一行数据已经通过约束检查，即使MAKE列也没有显示在父表AUTOMOBILES中，如下表：  http://www.uml.org.cn/sjjm/images/1178896651_ddvip_6277.png  表1 AUTOMOBILES  http://www.uml.org.cn/sjjm/images/1178896652_ddvip_9854.png  表2 INSURED\_AUTOS  延迟约束检验（Deferred Constraint Checking）  约束检验分两种情况，一种是在每一条语句结束后检验数据是否满足约束条件，这种检验称为立即约束检验（immediately checking）,另一种是在事务处理完成之后对数据进行检验称之为延迟约束检验。在缺省情况下Oracle约束检验是立即检验（immediately checking),如果不满足约束将先是一条错误信息，但用户可以通过SET CONSTRAINT语句选择延迟约束检验。语法如下：  SET CONSTRAINT constraint\_name|ALL DEFEERRED|IMMEDIATE --;  序列（Sequences）  Oracle序列是一个连续的数字生成器。序列常用于人为的关键字，或给数据行排序否则数据行是无序的。像约束一样，序列只存在于数据字典中。序列号可以被设置为上升、下降，可以没有限制或重复使用直到一个限制值。创建序列使用SET SEQUENCE语句。  CREATE SEQUENCE [schema] sequence KEYWORD  KEYWORD包括下面的值：  http://www.uml.org.cn/sjjm/images/1178896653_ddvip_8363.png  删除序列使用DROP SEQUENCE语句。  DROP SEQUENCE sequence\_name  索引(INDEXES)  索引是一种可以提高查询性能的数据结构，在这一部分我们将讨论索引如何提高查询性能的。ORACLE提供了以下几种索引：  B-Tree、哈希（hash)、位图（bitmap)等索引类型；  基于原始表的索引；  基于函数的索引；  域（Domain）索引。  实际应用中主要是B-Tree索引和位图索引，所以我们将集中讨论这两种索引类型。  B-Tree索引  B-Tree索引是最普通的索引，缺省条件下建立的索引就是这种类型的索引。B-Tree索引可以是唯一或非唯一的，可以是单一的（基于一列）或连接的（多列）。B-Tree索引在检索高基数数据列（高基数数据列是指该列有很多不同的值）时提供了最好的性能。对于取出较小的数据B-Tree索引比全表检索提供了更有效的方法。但当检查的范围超过表的10%时就不能提高取回数据的性能。正如名字所暗示的那样，B-Tree索引是基于二元树的，由枝干块（branch block)和树叶块（leaf block)组成,枝干块包含了索引列（关键字）和另一索引的地址。树叶块包含了关键字和给表中每个匹配行的ROWID。  位图索引  位图索引主要用于决策支持系统或静态数据，不支持行级锁定。位图索引可以是简单的（单列）也可以是连接的（多列），但在实践中绝大多数是简单的。位图索引最好用于低到中群集（cardinality)列，在这些列上多位图索引可以与AND或OR操作符结合使用。位图索引使用位图作为键值，对于表中的每一数据行位图包含了TRUE（1）、FALSE（0）、或NULL值。位图索引的位图存放在B-Tree结构的页节点中。B-Tree结构使查找位图非常方便和快速。另外，位图以一种压缩格式存放，因此占用的磁盘空间比B-Tree索引要小得多。 | |