DBS211

Week 5 - Data Definition Language (DDL)

Table of Contents

- Welcome
- Data Types
- Constraints
- CREATE
- ALTER
- DROP
- Temporary Tables
- Summary

Reading Materials

- Text - Chapter 15 (SQL Structured Query Language)
- Oracle Docs on Data Types

Welcome to Week 5

Welcome to Week 5 and a shift in the SQL we are learning. We are now going to dive a little deeper into DDL, Data Definition Language, and start to write SQL code to create physical tables in a database given the design specifications. We will learn the design of the specifications in the 2nd half of this course. After completing this week, you will be able to:

- write the SQL code to create tables given design specifications
- be able to choose the appropriate data type for different scenarios
- list and describe the 7 constaints used in databases
- write code to incorporate the constaints into tables at both creation time and after they are already created
- be able to create temporary tables and describe several primary uses for temporary tables in addition to understanding the pitfalls of temporary tables.

Data Types

Data types are an important topic in database design, but is somewhat controversal with resepct to standards, efficiencies and personal opinions. We will try and give you as many facts as we can such that you can decide the best data type to use in various scenarios. This choice is alse greatly hampered by the difference in data types between different DBMSs and their inconsistent adoption of ISO standards.

For this content, we will concentrate on data types for Oracle and SQL Server. We will cover both in this case, because of the great difference between the 2 and the implementation of standards and consistency. You will be using the Oracle version for this course, but it is important that database developers are familiar with the standards in all DBMSs.

In the chart below are the following notation:

- p - precision, the total number of digits
- s - scale, number of digits right of the decimal place
- n - size, numeric value

Chart References:

Oracle - https://docs.oracle.com/cd/B28359_01/server.111/b28318/datatype.htm#CNCPT012

SQL Server - https://docs.microsoft.com/en-us/sql/t-sql/data-types/data-types-transact-sql?view=sql-server-ver15

Variable Type	**Oracle**	**SQL Server**	**Size**	**Storage**	**Description**
Numerics					
Integer	number(3)	TinyInt	0 to 255	1 Byte	1 Byte storage
Integer	number(5)	SmallInt	-32,768 to 32,767	2 Bytes	2 Bytes storage
Integer	Int	Int	-2,147,483,648 to 2,147,483,647	4 Bytes	4 Bytes storage
Integer	number(19)	BigInt	2^63 (-9,223,372,036,854,775,808) to 2^63-1 (9,223,372,036,854,775,807)	8 Bytes	8 Bytes storage
Decimal	shortdecimal	Decimal(p,s)	- 10^38 +1 through 10^38 - 1	Precision/Storage 1-9, 5 Bytes 10-19, 9 bytes 20-28, 13 Bytes 29-38, 17 Bytes	Fixed Precision and Scale numbers. Precision Storage bytes 1 - 9 5 10-19 9 20-28 13 29-38 17
Decimal		Numeric(p,s)	- 10^38 +1 through 10^38 - 1		Fixed Precision and Scale numbers.
Decimal		SmallMoney	−214,478.3648 to 214,478.3647	4 Bytes	
Decimal		Money	-922,337,203,685,477.5808 to 922,337,203,685,477.5807	8 Bytes	Used for currency

Variable Type	Oracle	SQL Server	Size	Storage	Description
Decimal/Int	Number(p,s)		-1 x 10-130 to 1 x 10-125 up to 38 SigDig		
Decimal	Decimal	Float(n)	- 1.79E+308 to -2.23E-308, 0 and 2.23E-308 to 1.79E+308	n=1-24, 4 Bytes n=25-53, 8 Bytes	Approximate Numeric
Decimal		Numeric(p,s)			
Boolean		bit	0 or 1 only	1 Byte for each 8 bit fields	
Strings					
fixed length	char(n)	char(n)	8000 characters	n Bytes	fixed width string variable. Best used with strings that are always the same length. Phone numbers, serial numbers, GUIDs, Country Codes, etc.
variable length	varchar(n)	varchar(n)	8000 characters	n + 2 Bytes	variable length strings. Used for most string fields where the length changes. Examples: Names, Cities, Country Names, etc.
variable length		varchar(max)	2^31 characters	n + 2 bytes	

Variable Type	**Oracle**	**SQL Server**	**Size**	**Storage**	**Description**
variable length	varchar2(n)		4000 chacters (standard)	n bytes	
variable length	long	text	2,147,483,647 characters	n + 4 bytes	is a string field intended for large amounts of data. Examples: HTML Content, JSON Content, large descriptions, content management systems
Dates and Times					
	date	date	0001-01-01 to 999-12-31	3 Bytes	
		datetime	1753-01-01 to 9999-12-31	8 Bytes	
		smalldatetime	1900-01-01 to 2079-06-06	4 Bytes	
		time	00:00:00 to 23:59:59.99999	5 Bytes	
		datetime2	0001-01-01 to 9999-12-31	6 Bytes	
	timestamp				

The above data type list is not at all exhaustive, there are many specialty data types in each DBMS for specific reasons. For example, SQL Server has an image type that is specific for saving a binary stream of data such that it can be read, wrttien and changed easily through software.

When choosing the right data type for fields in the database, there are really only a few things to consider for beginners. Some of these are as follows:

Required	Choice
Fixed width strings	char(n)
Variable width strings	varchar(n)
Integers (up to 255)	number(3)
Integers (up to 32,000)	shortinteger
Integers (over 32,000 up to 2,000,000,000)	integer
Decimals	number(p,s)
Dates	Date
Booleans	number(1)

Constraints

In the business world, there are many rules surrounding business and the workflow of data. Constraints are used to assist in controlling how data is stored and place limitations, security and ensure consistency of data. We have already discussed a few of the constraints before, but in this module we will visit all 7 constraints in depth and show you how they are implemented at both creation time and afterwards.

The 7 constraint are:

Constraint	Description	Notes
Primary Key	The field are fields used to uniquely identify individual rows in a table.	Every table must have a single primary key. A Primary Key can be made up of more than one field (called a composite Key) By default, a PK is unique and an index in Oracle.

Constraint	Description	Notes
Foreign Key	A Key that enforces referential integrity between the child record and the parent table. The FK is placed on the child record table.	Foreign Keys can have cascading options as described in week 2.
Required	Fields that are not allowed to have NULL values	Uses the NOT NULL syntax
Unique	Requires every value in a column to be different.	Examples of fields where this can be used would include: email address, phone number, SIN, login or username.
Default Value	If a value is commonly used, then a default can be setup to ease the amount of effort in entering data.	Example: Kids playing in a city soccer league will almost all likely have the same city in their address, so make it the default value.
Check Range	Range constraints allow the developer to place limits or acceptable ranges on the data being entered as a value.	Example: A students grade must have a value between 0 and 100. No other values will be permitted.
Index	Indexes have various forms, but a generally used to pre-sort the data by the chosen field(s) such that search and sorting records will be more efficient.	Example: The phone book is typically indexed by last name. Logins, email addresses and other unique fields are often good candidates for indexing.

Typically, there are different ways to implement these constraints, as will be shown in the next module, but it is recommended that methods that allow naming of the constraints to be preferred. Constraints are objects of their own in the database and therefore can be added, changed and removed independantly of thier associated tables. Therefore, they are named regardless if the developer chose a name or not. When it is time to drop or change existing constraints, it is easiest done by name. However, if the name is unknown and was automatically generated, it makes this process much more difficult. By naming, the constraints, the developer knows the name and can easily make the alterations required.

CREATE

Here we are at the heart of DDL and where we need to start with writing SQL to create tables. The basic syntax of the CREATE TABLE statement is a little complex for those new to SQL but we will get you through it.

```
CREATE TABLE <tablename> (
    <fieldname_1> <datatype> <constraints>,
    <fieldname_2> <datatype> <constraints>,
....
    <fieldname_n> <datatype> <constraints>,
    CONSTRAINT <constraintname> <constraintType> <constaint parameters>
);
```

So lets examine the script we used last week for our samples and create the players table first. Here are the design specifications we will use, followed by the SQL script to create the table.

TABLE: players

FieldName	Type	Size	Required	PK/FK	Other
playerid	integer		✓	✓	
firstname	string	20	✓		
lastname	string	20	✓		
teamid	integer				

```
DROP TABLE players;   -- run this command first if you still have the players table in your database from last week.
```

```
CREATE TABLE players (
playerID    INT             PRIMARY KEY,
firstName   VARCHAR(20)     NOT NULL,
lastName    VARCHAR(20)     NOT NULL,
teamID      INT
);
```

Some things to note above:

- The required constraint can be implemented by using NOT NULL after the data type on the appropriate lines
- string fields of length 20, but names are variable in length, therefore we choose varchar as the data type
- There are 2 ways to add the primary key constraint, this way is the easiest and most common, however it will only work for single-field primary keys. For composite keys, the alternative method must be used.

Let us examine

TABLE: teams

FieldName	Type	Size	Required	PK/FK	Other
teamid	integer		✓	✓	
teamname	string	15	✓		
maxPlayers	int		✓		default 0, range from 0 to 25
shirtcolor	string	20			
homeField	string	15			

```
DROP TABLE teams;   -- run this command first if you still have the teams table in your database from last week.


CREATE TABLE teams (
teamID      INT         PRIMARY KEY,
teamName    VARCHAR(15) NOT NULL,
maxPlayers  INT DEFAULT 0,
shirtColor  VARCHAR(10),
homeField   VARCHAR(15),
CONSTRAINT maxPlayer_chk CHECK (maxPlayers BETWEEN 0 AND 25)
);
```

Notes:

- The CHECK constraint could have been added at the end of the MaxPlayers row, but then the constraint can't be named. Therefore this is the best way to add the constraint in the CREATE TABEL statement.
- the CHECK constraint name uses descriptive text and a suffix of _chk to indicate the constraint type. This is a standard naming convention.
- The DEFAULT constraint was added inline for maxPlayers. PRIMARY KEY, DEFAULT and NOT NULL constraints do not change often, and therefore do not really need to be named.

TABLE: fields

FieldName	Type	Size	Required	PK/FK	Other
fieldname	string	15	✓	✓	
address	string	50	✓		
manager	string	25			

```
CREATE TABLE fields (
fieldname   VARCHAR(15),
Address     VARCHAR(50),
Manager     VARCHAR(25),
PRIMARY KEY(fieldname)
);
```

In this 3rd example, we chose to add the PRIMARY KEY constraint as separate line. This is the method that must be used for composite keys. Example: `PRIMARY KEY(field1, field2)`

Adding a Foreign Key in CREATE statement

Notice that we did not add any foreign keys that will enforce referential integrity between realted fields. Let us now recreation the teams table and add a foreign key to the fields tables at creation time.

```
DROP TABLE teams;   -- run this command first if you still have the teams table in your database from previous work.


CREATE TABLE teams (
teamID      INT         PRIMARY KEY,
teamName    VARCHAR(15) NOT NULL,
maxPlayers  INT DEFAULT 0,
shirtColor  VARCHAR(10),
homeField   VARCHAR(15),
CONSTRAINT maxPlayer_chk CHECK (maxPlayers BETWEEN 0 AND 25),
CONSTRAINT team_field_fk FOREIGN KEY (homefield) REFERENCES fields(fieldname)
);
```

Notes:

- The name of the constraint is a mix of the two tables and an fk suffix.
- the first field in brackets is the child field in the current table to which the foreign key will be applied
- after the REFERENCES keyword, we must use the parent table name and in bracket provide the field in the parent table that the child references.
- you can define a FK inline, but again it is typically done as it's own line.

ALTER

So in the last example where we had to add the foreign key to the teams table, we dropped the table and then recreated it. This is okay when you are first creating a new blank database, but this is not possible if there is already data in the table without losing the data. Therefore, there must be a way to change the table after it has already been created, without dropping it. This is where the ALTER statement comes in.

The ALTER statement has many syntaxes as it is used for many things, so we will simply provide a variety of examples to give you the general form of the statement. Google is always a good friend when trying to figure out how to do specific things.

Example: We need to add a Foreign Key, to enforce referential integrity, to the teamid field in the players table such that we can not accidently put a player on a team that does not exist, and can not delete a team after a player has already been assigned to it.

```
ALTER TABLE players
    ADD CONSTRAINT player_teams_fk FOREIGN KEY (teamID) REFERENCES teams(teamID);
```

As you can see, the syntax for the FK constraint is almost identical to the previous version in the CREATE statement, with the addition of the word ADD in front of it. We then just need to use the ALTER part of the statement to know which table or object is being altered.

Adding a New Column

If we needed to add a new column to a table after it already exists without losing any data, we could use the ALTER statement for that. Let us add a field for date of birth for the players.

```
ALTER TABLE players
    ADD date_of_birth DATE;  -- could add constraints here too if needed, or use ALTER to add them after.
```

Dropping a Column

```
ALTER TABLE players
    DROP COLUMN date_of_birth;
```

Dropping a Constaint

```
ALTER TABLE players
    DROP CONSTRAINT player_teams_fk;
```

Note, dropping a constraint is hard to do if you do not know the name of the constraint.

DROP

The DROP statement is a powerful tool while developing a database and very dangerous in a live production databse. Use Caution.

To DROP an object from the database use the syntax:

```
DROP objecttype objectname;
```

To drop the fields table, we would use the command:

```
DROP TABLE fields;
```

Drop tables can also be affected by referential integrity, if we had player data in the players table that referred to

If a table that you are dropping has constraints on it, be careful to not break data integrity and leave orphaned records. However, if you ultimately are deleting the table, then you should use the following statement to drop both the table and it's associated constraint(s).

```
DROP TABLE <tablename> Cascade Constraints;
```

Temporary Tables

Temporary tables are an important part of database management and software development. Although they break most design rules of database theory, they play an important role. Some of the reasons for temporary tables include:

- Server Migration - When migrating a database from an old version to a new design, the data needs to be translated into the new schema, this requires the creation of temp tables,
- Data Import - Importing data from various external sources is often done using temporary tables, then SQL statements can be used to properly format the data and prepare it to be inserted into the live production tables
- Data Export - It is often important to export data from a database for data analysis, report generation, or data archival in larger databases.
- Fixed Data Storage - Often data calculaions on dataabase can be processor and memory intensive, and therefore can not be performed in real-time. Therefore, database designers create temporary static tables in the database to feed high traffic website and mobile applications saving the database server many cycles of calculations on demand.

Creating Tables Based on Other Tables

Practice Exercises

Data Types

Constraints

Create, Alter, Drop

Temp Tables