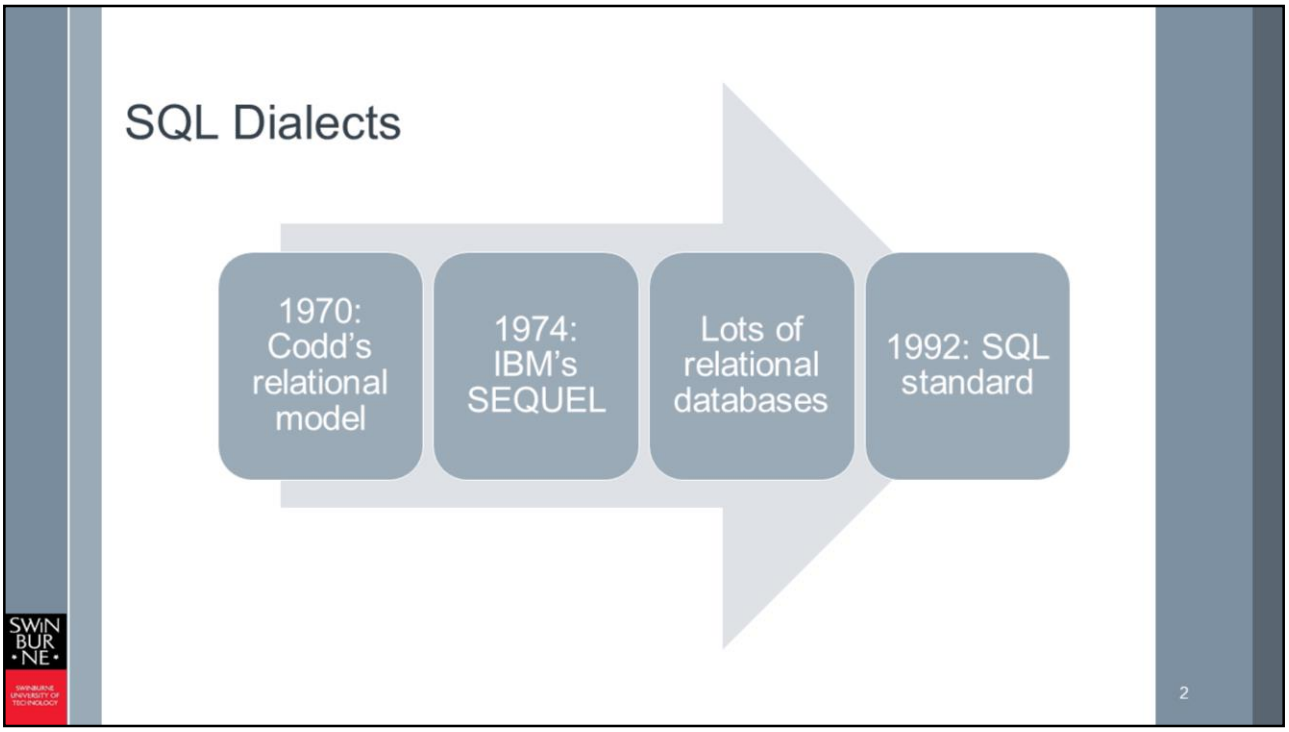
SQL

Data Definition Language (DDL)

SWIN BUR NE
SWINBURNE UNIVERSITY OF TECHNOLOGY

In this module we get started with SQL, which is a kind of descriptive language that lets us communicate with relational databases. SQL stands for structured query language. This presentation focuses on the DDL part of SQL, which is the one that lets you implement the data structures.



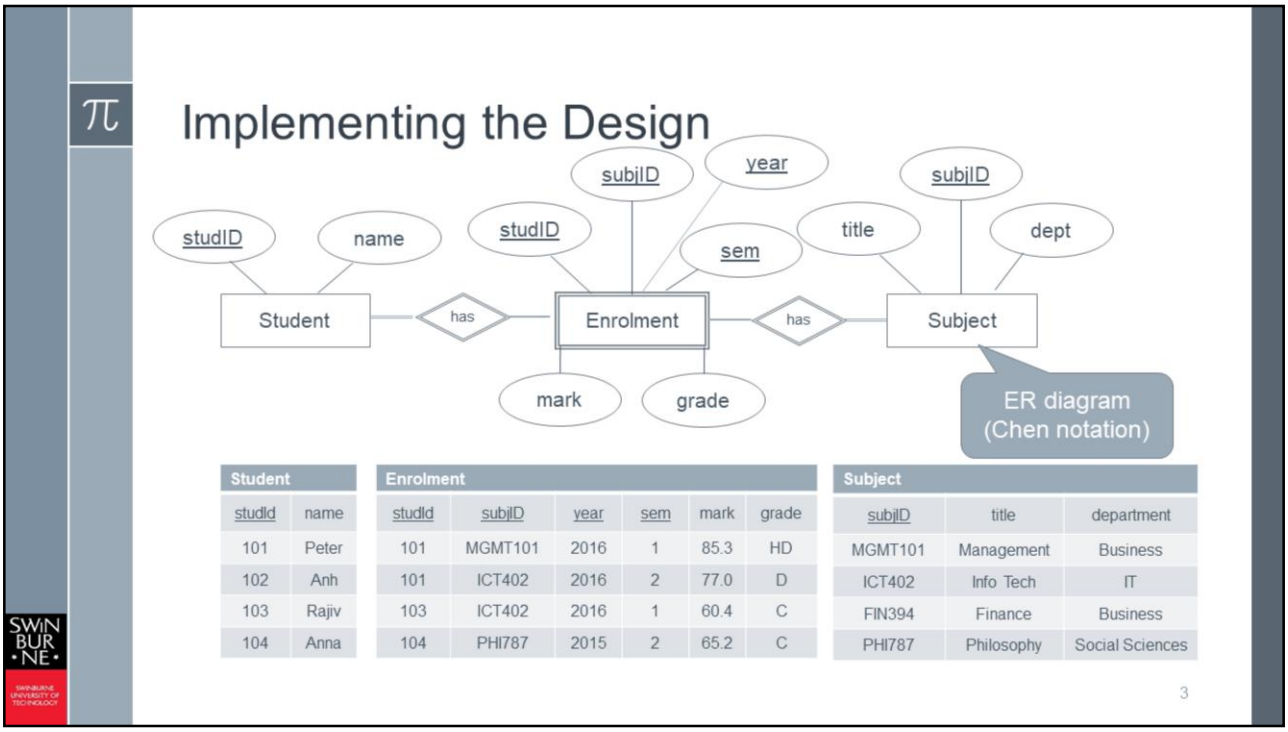
It all started with Edgar F. Codd publishing a paper called “A Relational Model of Data for Large Shared Data Banks” in 1970. His data model was highly structured, but also flexible enough to accommodate relationships between entities. These properties were very attractive in a time when electronic processing of data became accessible to mainstream companies. Until then, most companies did all of their bookkeeping on paper.

When the relational model was invented, IBM and Oracle immediately started working on implementations. It became clear that a scripting language was needed to access such databases. By 1974 IBM had produced the structured English query language, which is why many people still today mispronounce the name as SEQUEL. The name was changed to SQL in 1976 for legal reasons. SQL was standardised by the ISO first in 1989, but drastic changes were still made before the third version, the 1992 standard was published. There is also a 1999 standard, which contains additions that mainly deal with object-oriented additions to the model, but it has also added support for regular expressions. There is also SQL2003, SQL2008, SQL2011 – all standards that deal with SQL extensions. Essentially, all you need to know about SQL (and much more) is in the SQL-92 standard.

As you can see, lots of databases were developed way before the 1992 standard that cleaned up many inconsistencies in the SQL language. Oracle Corp got started on their database in the early seventies, and IBM's DB2 was developed at about the same time. Sybase is a product that started in 1984, and MS SQLServer was started in 1988 based on the Sybase code. So it comes as no surprise that none of them are 100% compliant with the SQL standard. The initial version of MySQL was published in 1995, but it isn't 100% compliant either. Depending on the features you use, you may have to change your embedded SQL if you migrate your data to another database product.

This phenomenon is often referred to as each database product having its own SQL dialect.

In these slides we use SQL according to the standard. You may have to adjust your syntax (and most likely your data types) when you run statements against an actual database.



Having designed the database relational model using an entity relationship diagram, it is time to implement the design on an instance of a database server.

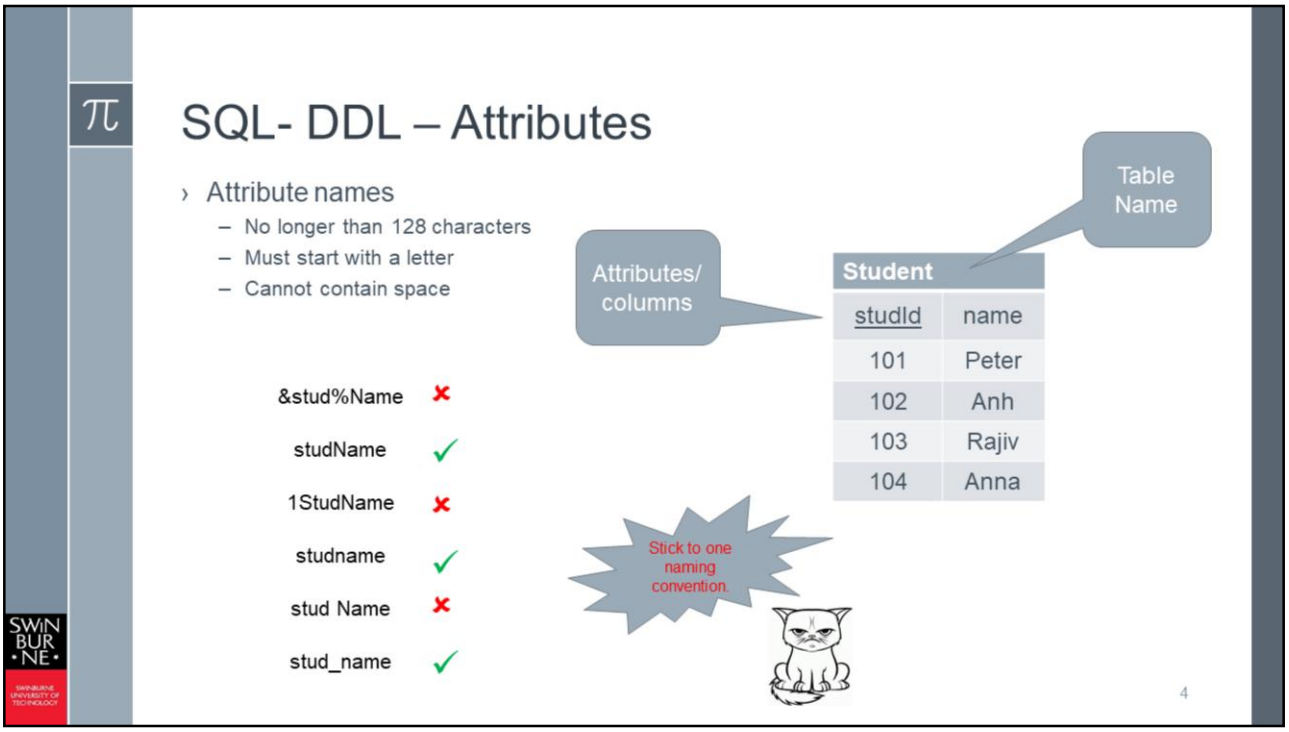
Each entity in the relational model becomes a table in the database system and each attribute becomes a column in the corresponding table.

If you are making a brand new database, you may want to create the database first. An instance of a database server can have several databases. If you have groups of entities that have nothing to do with each other (for example if you are hosting databases for different companies, or you have entirely separate applications such as a HR system and a sales management system), it is best to make a database for each of them.

Each database has its schema. A schema is a structural description of your database. So you can use ER diagrams or UML diagrams to describe the database schema.

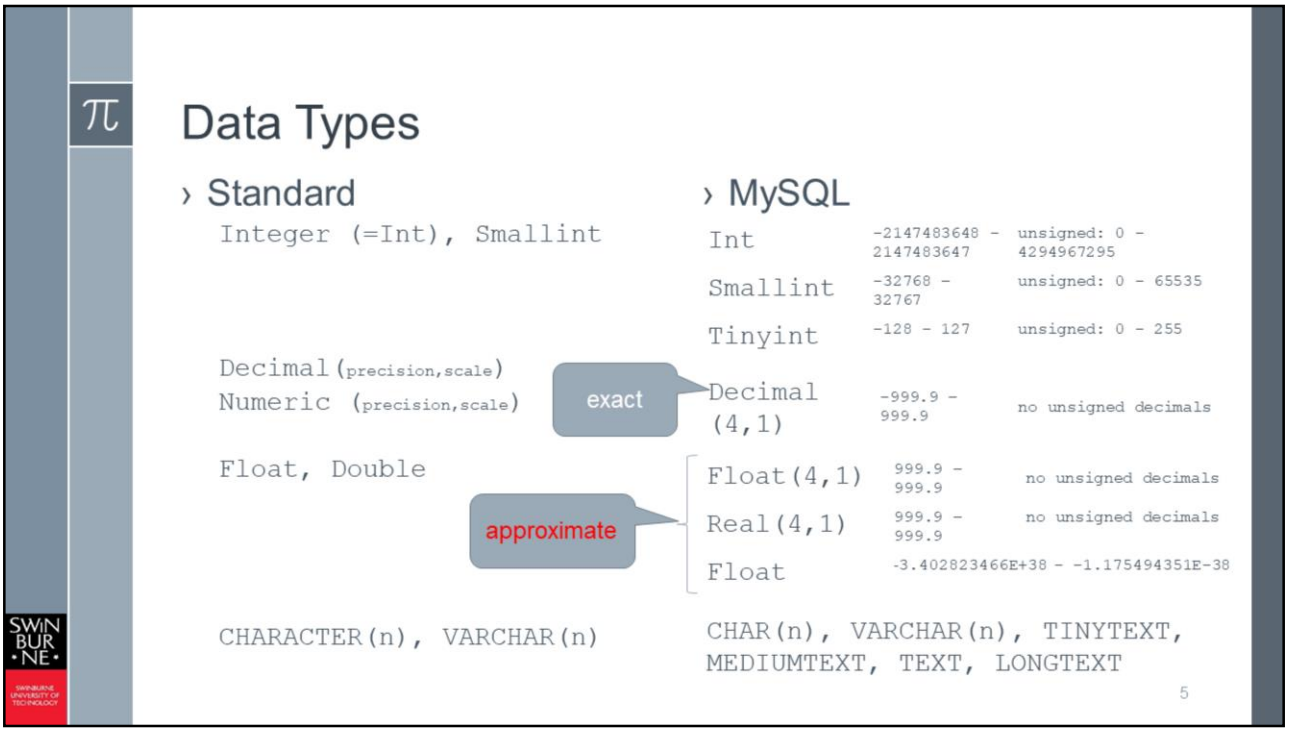
Within the database, we create the schema objects we need – schema objects are tables, indexes, views, triggers and so on.

The part of SQL which lets you define schema objects is called data definition language or DDL.



When you create a new table, you have to create its member attributes as well. Attributes need meaningful names so people understand what the attribute is about.
There are a few technical rules for naming attributes:
Attribute names must be strings of characters which consist of uppercase or lowercase letters, digits and underscores. Attribute names can't start with a number. You can't have spaces in attribute names.

Apart from these technical rules, you should make your own naming convention. Many people prefix a column name with the table name, like in the example. Older databases use underscores to connect two parts of a name. More recently, people have started using camel notation like in Java. You can use capital letters for column names, but this is more unusual. People often start the table names with capital letters, but the column names with a lower case letter. Whatever you do, stick to one style, don't mix or the database looks really messy.



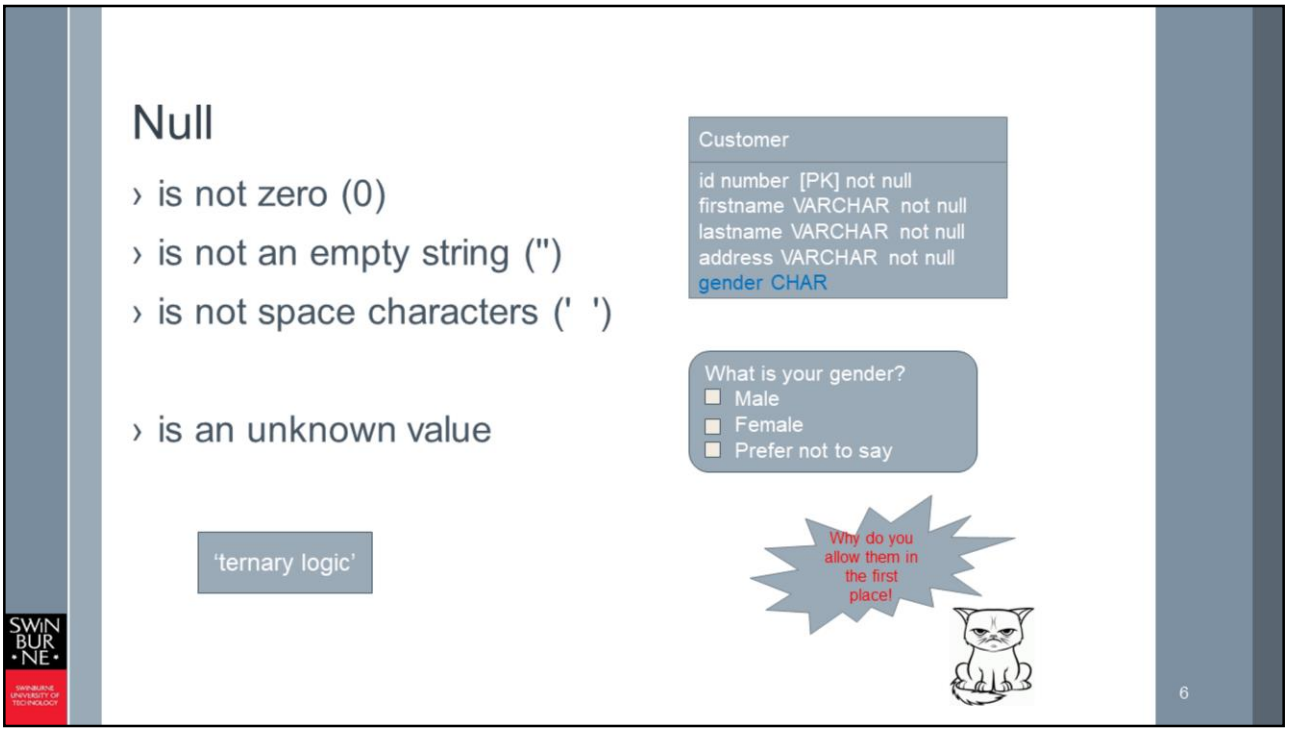
The SQL standard defines quite a large number of data types, many more than mentioned here. Each database product implements these slightly differently. So when you start implementing your database, investigate the data types of your database product first.

For example, MySQL has its own implementation of the standard types, but then it also offers types that are not mentioned in the standard. Sometimes these custom types work exactly the same as the types from the standard.

MySQL has at least double as many data types as the standard. On the slide there are only a few select ones. MySQL also has mediumint and bigint, and you can use Integer and specify the number of bytes that you want. There is also Numeric, and it works the same as decimal.
Trying to remember them makes little sense. What you should remember, though, is that some numbers will be imprecise. Because of the conversion to binary and the limited precision, these numbers, marked approximate, will not be 100% accurate all the time. If you need to express monetary values or other items where we cannot afford to have imprecise values, use decimal not float, real or double.

Another interesting fact is that with integers, you can have a signed and an unsigned version. If you specify nothing, the default is signed. Signed means

the first bit in the number is used to signify whether the number is positive or negative. If you know exactly that your number can never be negative, it makes a lot of sense defining it as unsigned. Most databases can do unsigned, so look into this as well.



When you create a database, you'll have attributes where you cannot guarantee that you'll always know the value for them. This kind of 'empty value' is called a null value.

Null values often cause headaches in SQL queries. They give rise to 'ternary logic'. Binary logic is true false. Ternary logic is true, false and unknown. But we'll get into this when we talk about queries.

Null means we don't know the value yet. Perhaps we'll never know it. Perhaps we just haven't asked yet.

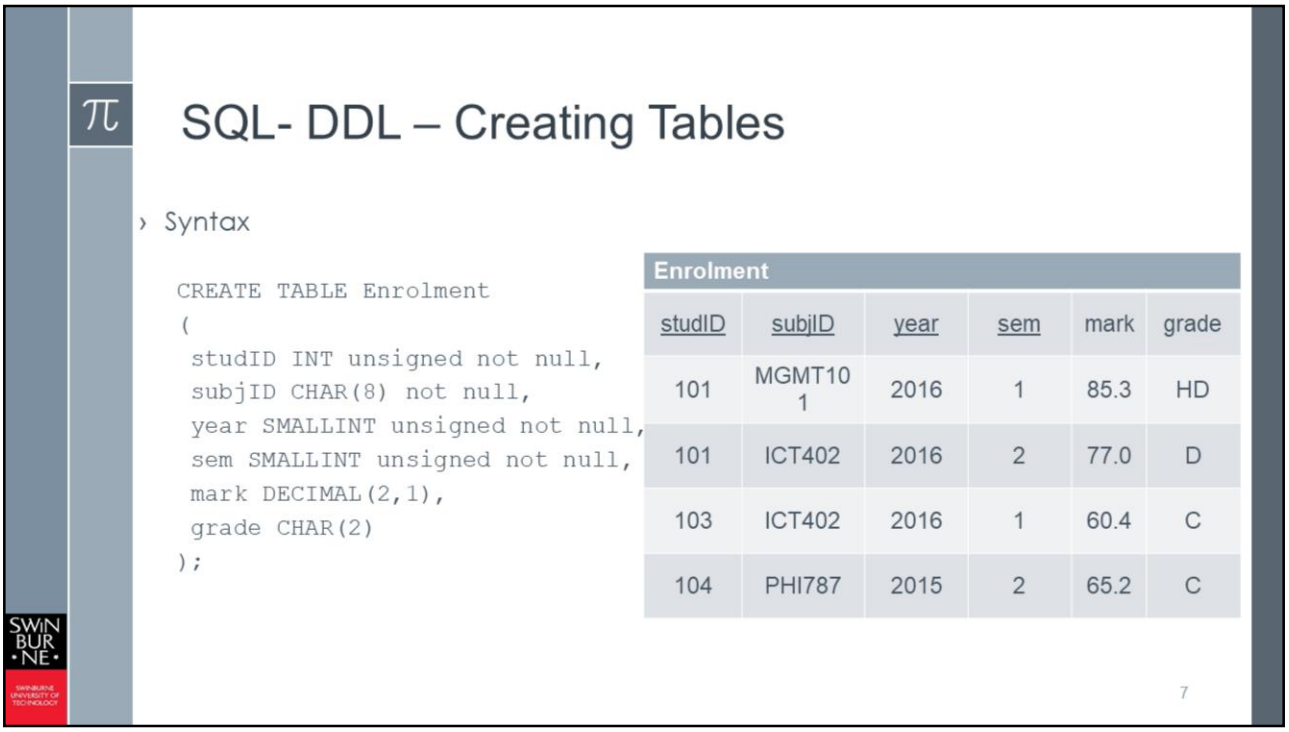
There are many situations where you can't be sure your database will ever have a value for a field. You might think let's just exclude such fields, but this isn't always an option. In the example, we have a gender field in the customer table. Many companies like to know the gender of their customers for targeted marketing. But a company cannot force its customers to reveal their gender.

We could decide to exclude attributes that are not guaranteed to have values, but this may be counterproductive – for example, most people will volunteer their gender without even thinking, and you would miss out on the information if you abolished the gender field.

You could say the gender field should have a default value, but this has other problems. If we say the default is female, for any field that has a value of female we can't actually say whether this value stems from a customer's answer or was made up by the DBMS because there was no value.

Consequently, relational DBMSs allow null values, and people use them all the

time. But null values are problematic in some queries, and we'll look into this later. For now all you have to know is that you should make as many fields not nullable as possible.



Now that we know the basics about naming and typing attributes, as well as null values, we can look at the SQL syntax for creating a table.

The syntax is easy enough. The data types are trickier. You have to decide whether a field is large enough to capture all possible values. Looking at the numbers for studID, we would think that Smallint would be more than enough. But we may enrol lots of students over the years, so it may be safer to have an integer. Chances are that int is only four Bytes longer, and we don't worry about storage space on this scale. What we know for sure is that negative values are out of the question for student IDs, so we make the number unsigned. This doubles the number of possible entries, so Smallint might actually be long enough.

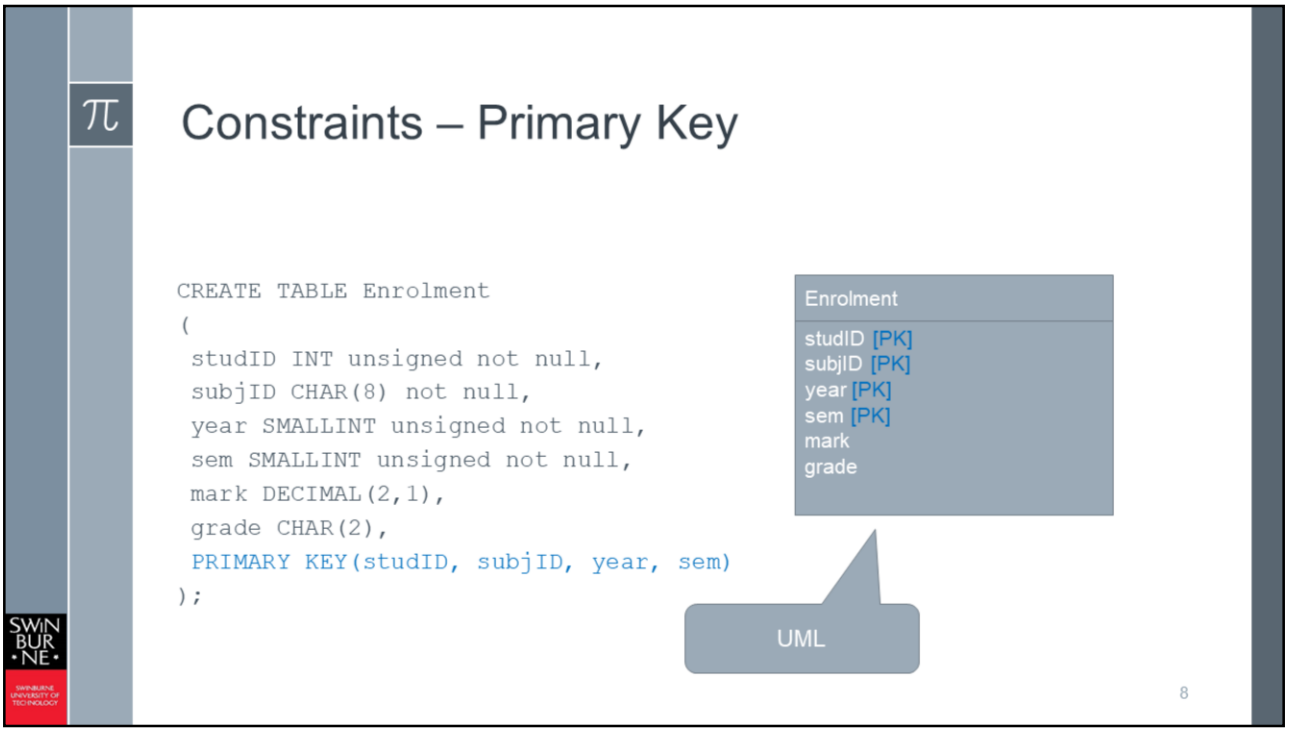
If you use the MySQL data types, you may want to use and integer and specify the number of Bytes it should use.

subjID is also an ID field, but in this case it contains letters, so all we can do is make it character data. Because the variation in length is really insignificant, we might as well make it char not varchar.
Semester and year are fields that are numeric but you'll never use them for calculations. So we might choose to make them character strings.

The mark field is clearly a decimal value with a scale of one (one digit after the dot). If we use decimal in MySQL, we know there won't be any floating point errors. For any other database you would have to check this.

Grade is clearly a character string of length 2. Using varchar is not very meaningful, because the values are either one or two characters long, so saving space is not a great advantage over having a steady length field.

We can safely say that studID, subjID, year and semester are compulsory values that must be known from the start. There is no enrolment if we don't know who the student is, what subject we are enrolling her in and for what semester. The mark and grade are different – in the beginning of each semester they will be unknown. Therefore these two must be nullable.



The table we created just now is perfectly usable – you could begin to insert data into it. But you may remember that some fields in the ER diagram were specified as primary keys. In the case of Enrolment, we had a composite primary key that included studID, subjID, year and semester.
The Create table statement should always contain a primary key constraint. Although id fields can be used without imposing a primary key constraint, the constraint ensures that there will be no duplicate values. Primary key constraints will also make sure you specify the not null with the column.

If you don't add the constraint, something might go wrong and you might have duplicate entries. This is a problem when you expect a field to be a unique identifier. All of a sudden you might have a student enrolled in the same subject twice in the same semester. When you run a count over a students enrolments, the numbers won't match up.

Doing a lazy job when defining data structures usually results in lots of detective work down the track and days spent sorting out discrepancies as well as having to explain the problem to your boss.

Constraints – Foreign Key

```
CREATE TABLE Enrolment
(
 studID INT unsigned not null,
 subjID CHAR(8) not null,
 year SMALLINT unsigned not null,
 sem SMALLINT unsigned not null,
 mark DECIMAL(2,1),
 grade CHAR(2),
 PRIMARY KEY(studID, subjID, year, sem)
 FOREIGN KEY(studID) REFERENCES Student(studID),
 FOREIGN KEY(subjID) REFERENCES Subject(subjID)
);
```

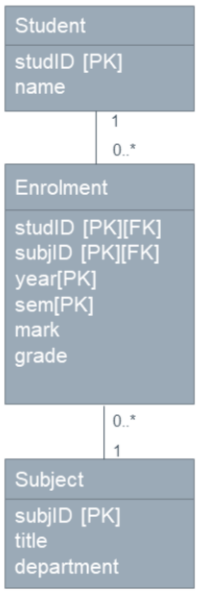
Similar to the primary key constraint, if you impose the foreign key reference in the table implementation, the DBMS makes sure you cannot state a value that does not exist as a primary key in the referenced table. This ensures that there are no orphaned rows in the child table (the table that has the foreign key). But you have to be aware that if you allow null values in the foreign key column, the DBMS will allow nulls even if it is a foreign key column. This means that orphaned rows can still exist. However, any value other than null will be matched against a primary key in the parent table. Avoid null values with foreign keys if you can.

The relationship between tables exists even if there is no foreign key constraint. You can still match the row on the foreign key column. But like in the case of the primary key constraint, missing constraints always present a risk of database corruption. The DBMS cannot stop a wrong value from being inserted when there is no foreign key constraint.

Foreign keys ensure the referential integrity of the database is preserved.

Constraints – Check Constraint

```
CREATE TABLE Enrolment
(
 studID INT unsigned not null,
 subjID CHAR(8) not null,
 year SMALLINT unsigned not null,
 sem SMALLINT unsigned not null DEFAULT 1,
 mark DECIMAL(2,1) CHECK(mark
         >= 0.00 AND mark <= 100.0),
 grade CHAR(2),
 PRIMARY KEY(studID, subjID, year, sem)
 FOREIGN KEY(studID) REFERENCES Student(studID),
 FOREIGN KEY(subjID) REFERENCES Subject(subjID)
);
```



Check constraints let you specify allowed values for columns. You can also use unique constraints by putting the keyword UNIQUE behind an attribute declaration. This means although the column is not a primary key, the DBMS will take care that no duplicate values appear in this column.

The default keyword is not a constraint, rather a tool that lets you specify a value when we don't know the actual value yet or when most of the time, the attribute is set to a particular value. This is helpful when you don't know the actual value yet but don't want to allow nulls with the column. When you add rows to the table, you can omit the attribute with the default value and the DBMS will apply the default. This is a good way of avoiding null values, but it works only if there is a meaningful value that can be the default.

SQL- DDL – Making Changes

› In table Student, add student e-mail address:

```
ALTER TABLE STUDENT
ADD COLUMN email VARCHAR(20);
```

› ..and remove it again:

```
ALTER TABLE STUDENT
DROP COLUMN email;
```

› Adding constraints:

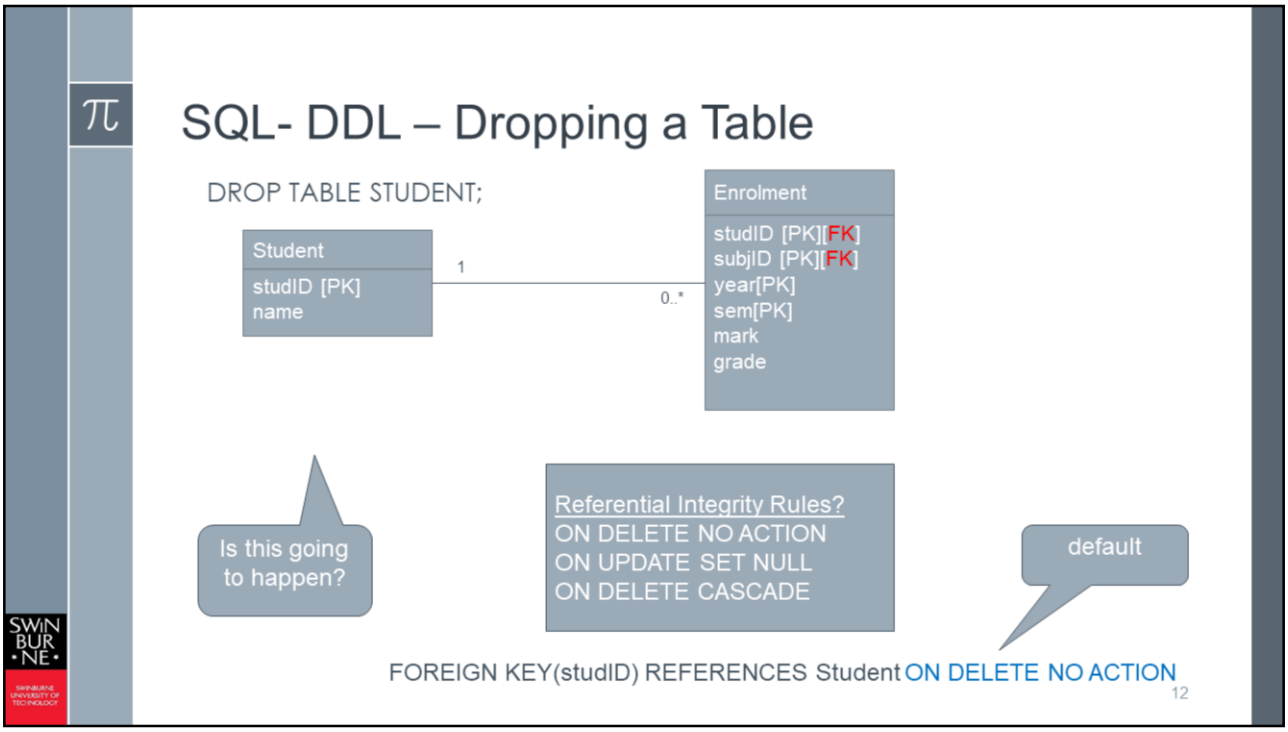
```
ALTER TABLE STUDENT
ADD CONSTRAINT studID_const PRIMARY KEY (studID);
```

Student

studID	name	email
101	Peter	peter@company.com

11

The ALTER command lets you make changes to existing tables. Students tend to avoid it because in a practice environment, you have the option of eliminating and recreating tables. Production databases don't have this advantage. If you need to add or remove another column, you need ALTER table. Using ALTER TABLE you can also add primary key and other constraints.
The example shows the standard syntax, which gives the constraint a proper name. MySQL also has a shorter version without the constraint name.



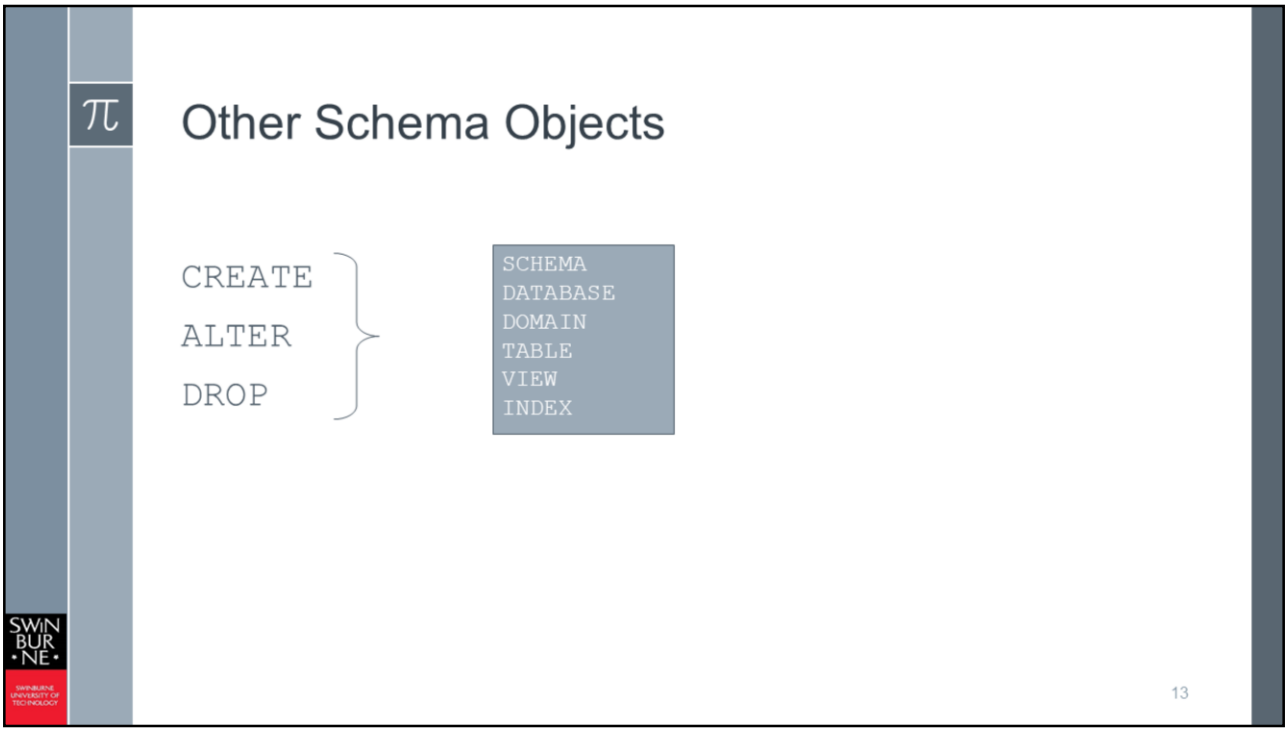
If you need to eliminate a table from your schema, you need to use the DROP command.
Using the drop statement is a serious matter in real life since once a table is dropped it cannot be recovered, therefore, you must make sure that it is safe to do this. DBMSs won't even ask you whether you're sure you want to do this. The table just disappears. Having said that, if the database was set up properly including the appropriate user rights, you shouldn't be able to do this as a normal DB user.

In the case of the student table, it's actually quite likely nothing at all will happen when you try to delete it. You'll get some kind of error message that you may or may not be able to make sense of at this stage.
The issue here is that there are foreign key references in the enrolment table, which is the child table that points to the parent table Student. Now if you were sloppy enough not to put in any foreign key constraint, this will make no difference whatsoever. The table will just disappear and we have lots of orphaned rows in the enrolment table.

If you did implement the foreign key constraint, chances are the delete won't happen, although this depends on the database product. More precisely, it depends on the referential integrity rules applied. Most users don't specify referential integrity rules when implementing a foreign key constraint. This only

means that the DBMS applies the default, which, in most cases, is ‘on delete no action’. Referential integrity rules actually specify what should happen when you delete a row in the parent table – which is the student table here. ON DELETE NO ACTION means the deletion of the parent row will be stopped if the row has child rows in Enrolment. ON DELETE CASCADE is the most dangerous – it means all child rows disappear as well! This is a very dangerous rule, especially if the child table has other child tables. But if the child of the child specifies ON DELETE NO ACTION, the cascading will be stopped and nothing happens after all! In short, referential integrity rules seem simple but are actually quite complex in practice. That’s why you’re not expected to master them here – you’re only expected to remember there was such a concept when you are making your own databases. When you remember that these rules exist, you know how to explore the options for your database product.

Although the referential integrity rules is designed for deletes, in most database products it will also affect dropping the entire table. Students therefore tend to skip implementing foreign keys, which is not recommended. If you need to drop tables, you just have to drop the child table first. Dropping Enrolment and then Student won’t give you any problems. If this is not convenient, you can temporarily disable foreign key constraints in MySQL by setting the variable FOREIGN_KEY_CHECKS = 0;



Schema objects are objects that are part of a database. In fact, database and schema are often used interchangeably. Schema is the standard term. Most DBMSs let you create databases instead of or in addition to SCHEMAs.
DDL is the part of SQL that lets you define your database. So it comes as no surprise that the DDL statements CREATE, ALTER, DROP are used to manipulate schema objects. You know what a table is; a domain is a custom data type. If you have many columns that have the same data type and perhaps constraint, you may want to define a domain to make defining attributes easier. But this is not something you need to remember.
VIEWs are usually badly understood and underused, but we will have to talk about them later, because first you have to learn about queries. Indexes are structures that can help make queries faster, but we have to talk about them after discussing queries.

Summary

- SQL has two major components: Data Definition Language and Data Manipulation Language.
- Data Definition Language (DDL) has three major statements that let you define your database structure: CREATE, ALTER, DROP.
- When creating a table, each column must have a data type that defines the values that it could store.
- Don't skip implementing PRIMARY and FOREIGN KEY constraints.
- Dropping tables must be carefully planned.
- Dropping a table may not succeed when child rows reference its tuples. Dropping child tables first is one way to fix this. Disabling foreign key constraints is another.

14

Here are the most important points discussed in this module. You may want to stop the recording to have a read through them. When you are ready, start the quiz about this module.