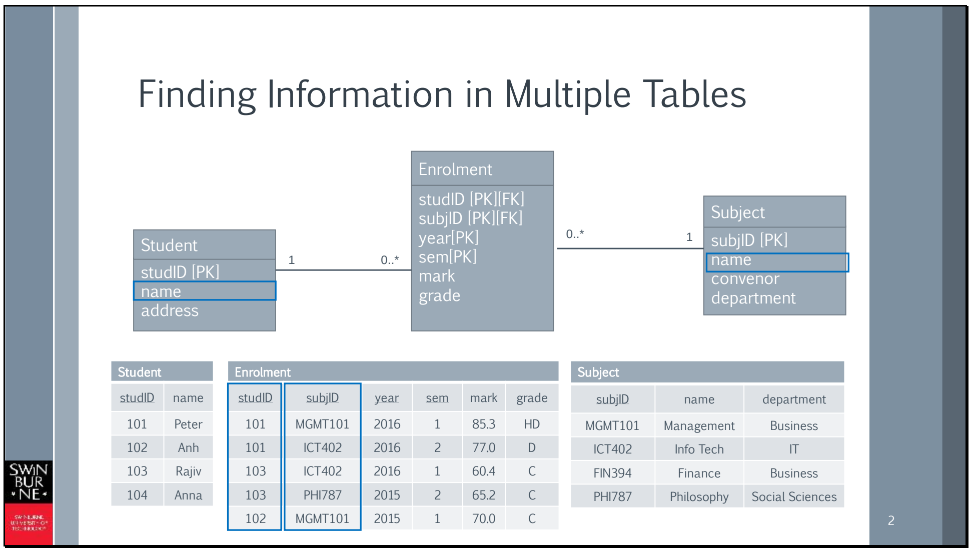
SQL DML

Querying Multiple Tables

In this presentation we’ll look at how to combine information from several tables.

Slide 2



Finding Information in Multiple Tables

Student
studID [PK]
name
address

1 0..*

Enrolment
studID [PK][FK]
subjID [PK][FK]
year[PK]
sem[PK]
mark
grade

0..* 1

Subject
subjID [PK]
name
convenor
department

Student

studID	name
101	Peter
102	Anh
103	Rajiv
104	Anna

Enrolment

studID	subjID	year	sem	mark	grade
101	MGMT101	2016	1	85.3	HD
101	ICT402	2016	2	77.0	D
103	ICT402	2016	1	60.4	C
103	PHI787	2015	2	65.2	C
102	MGMT101	2015	1	70.0	C

Subject

subjID	name	department
MGMT101	Management	Business
ICT402	Info Tech	IT
FIN394	Finance	Business
PHI787	Philosophy	Social Sciences

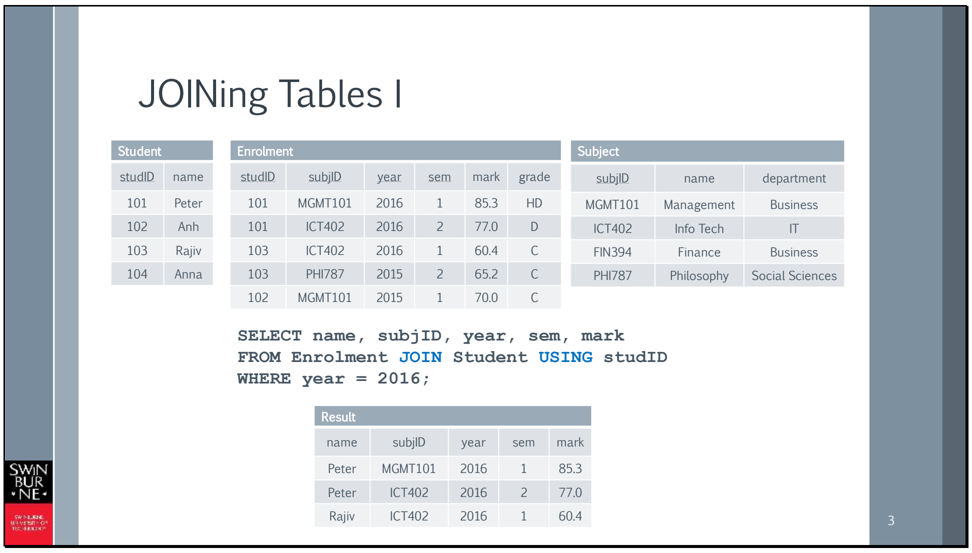
You have seen these three tables before. There are two strong entities, student and subject, and a weak entity between them. You may still be grappling with the concept of weak entities. If this is the case, think about where you could place the mark column if the enrolment didn’t exist – if it was in the student table, we would need as many duplicates for the student as she has had enrolments. Not very practical. If the mark was in the subject table, we would need as many duplicates for each subjects as there are students. This is why we need another table that holds all the properties of the combination of student and subject, which is naturally enrolment.

Most of the time, our databases have one-to-many relationships, as we do here. When we combine all rows from two tables with a one-to-many relationship, how many rows do we get? Naturally, as many as we have child rows – the child table is the one on the many end. The columns from the parent table (the table on the one end) just have lots of duplicate values. You might guess that looking at the foreign key column, because the foreign key column dictates what the entries in the parent table columns are.

Another thing is that tables often have attributes with the same name. You have been told that it is good practice to prefix an attribute with the table name. In many legacy databases, this rule hasn’t been observed. So you’ll have to deal with situations where different tables have attributes with the same names. In the example, both subject and student have a name column so we can see how to deal with this issue.

When we want to combine information from several tables, we talk about joining tables. Although what we mostly do is joining a subset of the tuples in the tables by a joint column.

Slide 3



JOINing Tables I

Student

studID	name
101	Peter
102	Anh
103	Rajiv
104	Anna

Enrolment

studID	subjID	year	sem	mark	grade
101	MGMT101	2016	1	85.3	HD
101	ICT402	2016	2	77.0	D
103	ICT402	2016	1	60.4	C
103	PHI787	2015	2	65.2	C
102	MGMT101	2015	1	70.0	C

Subject

subjID	name	department
MGMT101	Management	Business
ICT402	Info Tech	IT
FIN394	Finance	Business
PHI787	Philosophy	Social Sciences

```
SELECT name, subjID, year, sem, mark
FROM Enrolment JOIN Student USING studID
WHERE year = 2016;
```

Result

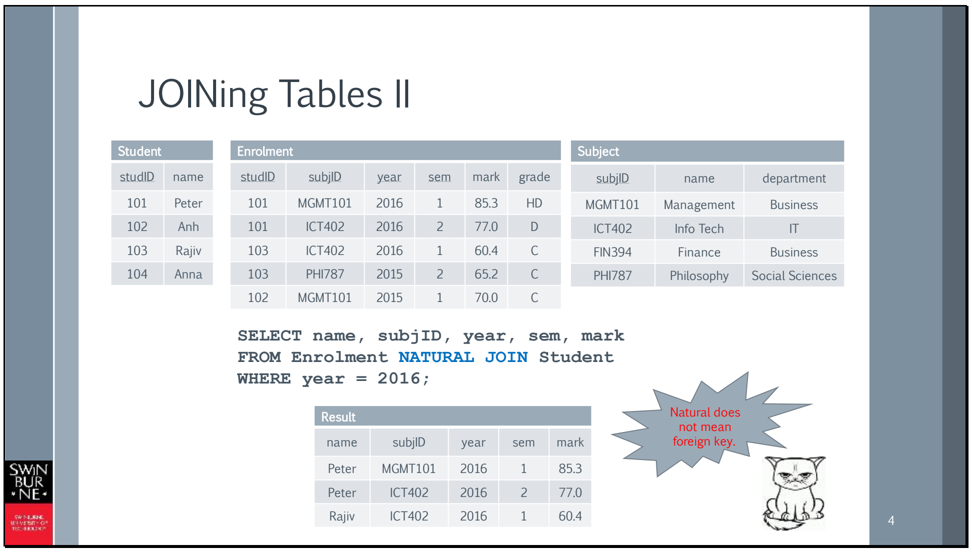
name	subjID	year	sem	mark
Peter	MGMT101	2016	1	85.3
Peter	ICT402	2016	2	77.0
Rajiv	ICT402	2016	1	60.4

3

Suppose we want to have a listing of students and their marks for the subjects they took in 2016. We need to project the student name, the subject, semester and mark. To get these from the two tables, we have to include both tables in the FROM clause of the statement. The result is what we expect – essentially the rows in the result set are based on the rows in the Enrolment table. The studID has been replaced by the student name, which is more informative.

There are several variations of syntax we can use to join the tables. JOIN and USING is one of them. USING works when you have a column that has the same name in both tables. The order of the tables makes no difference. You could also write FROM Student JOIN Enrolment.

Slide 4



JOINing Tables II

Student

studID	name
101	Peter
102	Anh
103	Rajiv
104	Anna

Enrolment

studID	subjID	year	sem	mark	grade
101	MGMT101	2016	1	85.3	HD
101	ICT402	2016	2	77.0	D
103	ICT402	2016	1	60.4	C
103	PHI787	2015	2	65.2	C
102	MGMT101	2015	1	70.0	C

Subject

subjID	name	department
MGMT101	Management	Business
ICT402	Info Tech	IT
FIN394	Finance	Business
PHI787	Philosophy	Social Sciences

```
SELECT name, subjID, year, sem, mark
FROM Enrolment NATURAL JOIN Student
WHERE year = 2016;
```

Result

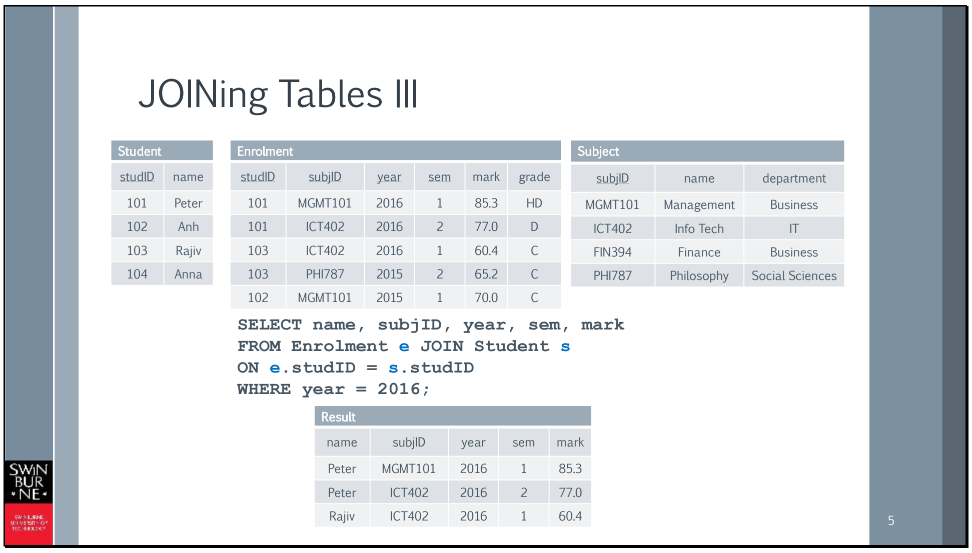
name	subjID	year	sem	mark
Peter	MGMT101	2016	1	85.3
Peter	ICT402	2016	2	77.0
Rajiv	ICT402	2016	1	60.4

Natural does not mean foreign key.

4

Another variation of syntax is NATURAL JOIN. Many people think that a natural join is the same as joining tables by foreign key. This is not the case. Natural joins use the columns that have the same name in both tables. This is important to remember – if you tried to join Student and Subject, natural join would use the name attribute. Apart from the fact that joining student and subject would never be meaningful, you might be surprised to see the result set is empty, because student names and subject names never have the same value.

Slide 5



JOINing Tables III

Student

studID	name
101	Peter
102	Anh
103	Rajiv
104	Anna

Enrolment

studID	subjID	year	sem	mark	grade
101	MGMT101	2016	1	85.3	HD
101	ICT402	2016	2	77.0	D
103	ICT402	2016	1	60.4	C
103	PHI787	2015	2	65.2	C
102	MGMT101	2015	1	70.0	C

Subject

subjID	name	department
MGMT101	Management	Business
ICT402	Info Tech	IT
FIN394	Finance	Business
PHI787	Philosophy	Social Sciences

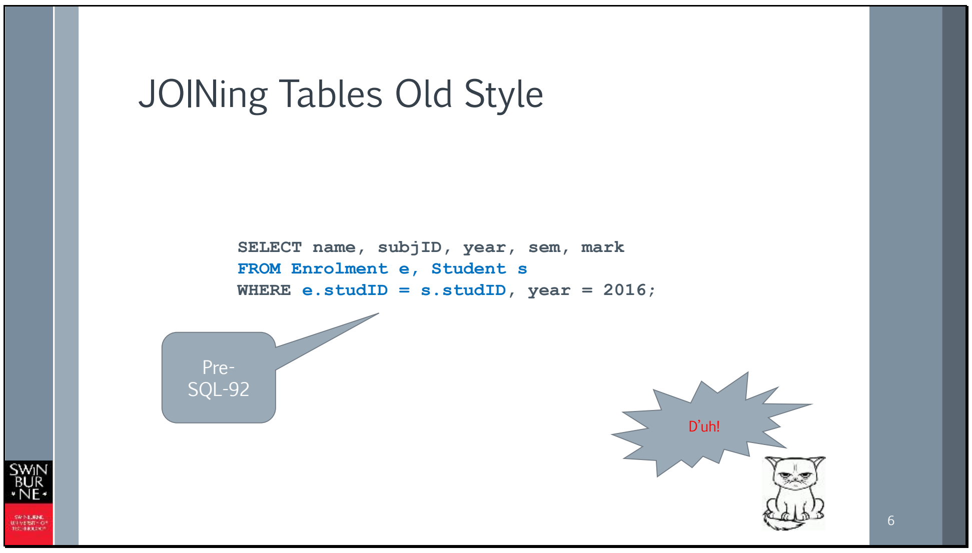
```
SELECT name, subjID, year, sem, mark
FROM Enrolment e JOIN Student s
ON e.studID = s.studID
WHERE year = 2016;
```

Result

name	subjID	year	sem	mark
Peter	MGMT101	2016	1	85.3
Peter	ICT402	2016	2	77.0
Rajiv	ICT402	2016	1	60.4

If you don’t have identical column names for the join column, this is what you do. Join conditions using ON let you specify a relationship between the columns. By far the most common is equijoin, where there is an equal sign between the columns. In theory you could have other operators, such as less than and greater than.
In this case, using the ON structure is overkill, because the columns actually have the same name. This adds another complication – if you just say ON studID = studID the DBMS will complain that your columns are ambiguous.
But there is a way out – you can give each table an alias that makes the clause shorter. Now the column names are not ambiguous and the clause is reasonably short.

Slide 6



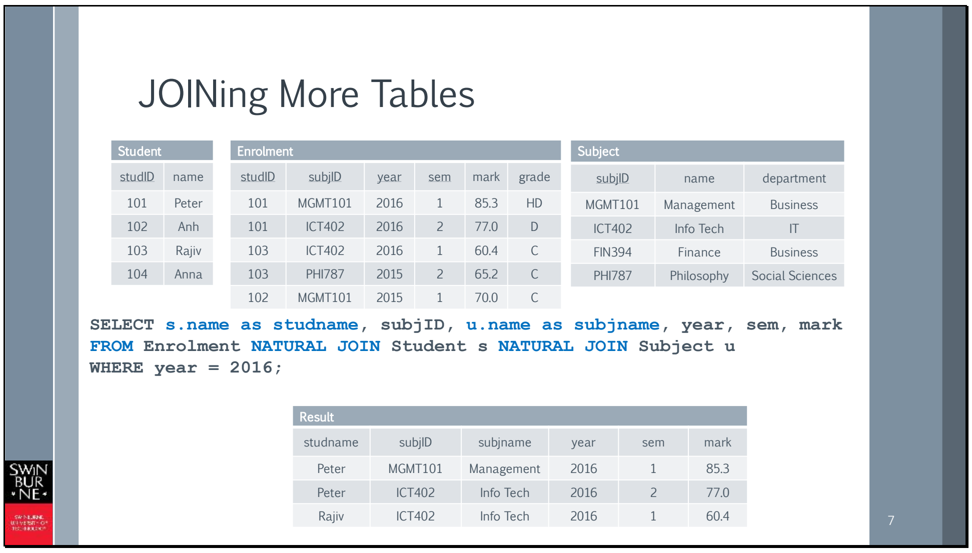
JOINing Tables Old Style

```
SELECT name, subjID, year, sem, mark
FROM Enrolment e, Student s
WHERE e.studID = s.studID, year = 2016;
```

Pre-
SQL-92
D'uh!
6

Before the SQL-92 standard was ratified, the JOIN keyword did not exist. All tables were just listed in the FROM clause, separated by commas.
There is no special keyword for specifying the join columns, so these are just added to the WHERE clause, which can make the WHERE clause very long and difficult to read. That's why this notation is discouraged. But you can't always avoid it – some database products have never caught up with the JOIN statement. Oracle, for example, needs this old style join syntax.
Having said that, even the old style join notation lets you use aliases for tables. But here is your pitfall – if you forget to use the matching column condition in the WHERE clause, you get the Cartesian product!

Slide 7



JOINing More Tables

Student

studID	name
101	Peter
102	Anh
103	Rajiv
104	Anna

Enrolment

studID	subjID	year	sem	mark	grade
101	MGMT101	2016	1	85.3	HD
101	ICT402	2016	2	77.0	D
103	ICT402	2016	1	60.4	C
103	PHI787	2015	2	65.2	C
102	MGMT101	2015	1	70.0	C

Subject

subjID	name	department
MGMT101	Management	Business
ICT402	Info Tech	IT
FIN394	Finance	Business
PHI787	Philosophy	Social Sciences

```
SELECT s.name as studname, subjID, u.name as subjname, year, sem, mark
FROM Enrolment NATURAL JOIN Student s NATURAL JOIN Subject u
WHERE year = 2016;
```

Result

studname	subjID	subjname	year	sem	mark
Peter	MGMT101	Management	2016	1	85.3
Peter	ICT402	Info Tech	2016	2	77.0
Rajiv	ICT402	Info Tech	2016	1	60.4

7

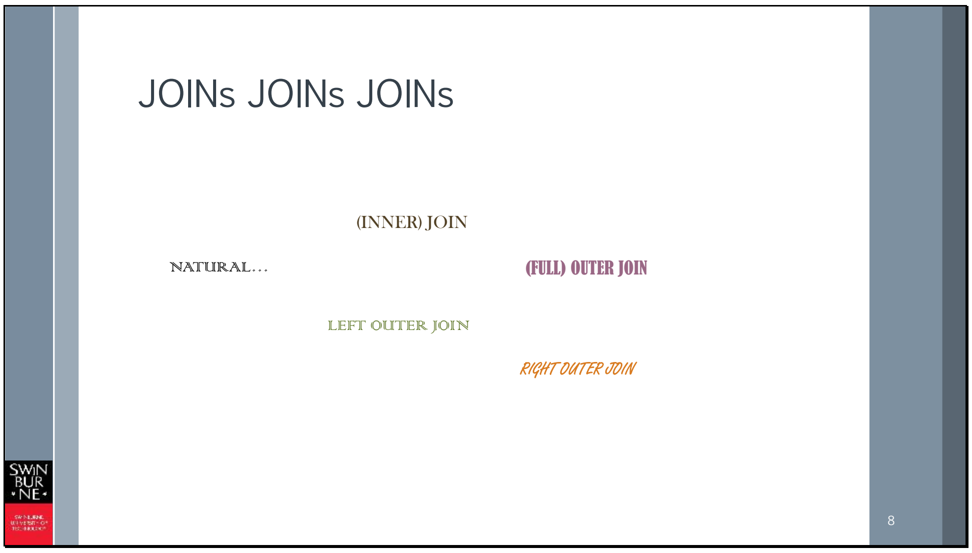
If you don't have identical column names for the join column, this is what you do. A Join clause that uses ON lets you specify the columns that are used for matching the rows.

In this case, using the ON clause is overkill, because the columns actually have the same name. The fact that they do have the same name adds another complication – if you just say ON studID = studID the DBMS will complain that your columns are ambiguous. So you have to prefix each column name with the table, or in this case, the alias.

Actually, you could just use a natural join and the query would be easier to read.

There is another issue here. Both student and subject have a name. In the projection, we have to prefix the name field. The fact that we are giving the name field an alias is our own choice, it is not necessary, just useful for clarity.

Slide 8



JOINs JOINs JOINs

(INNER) JOIN

NATURAL…

(FULL) OUTER JOIN

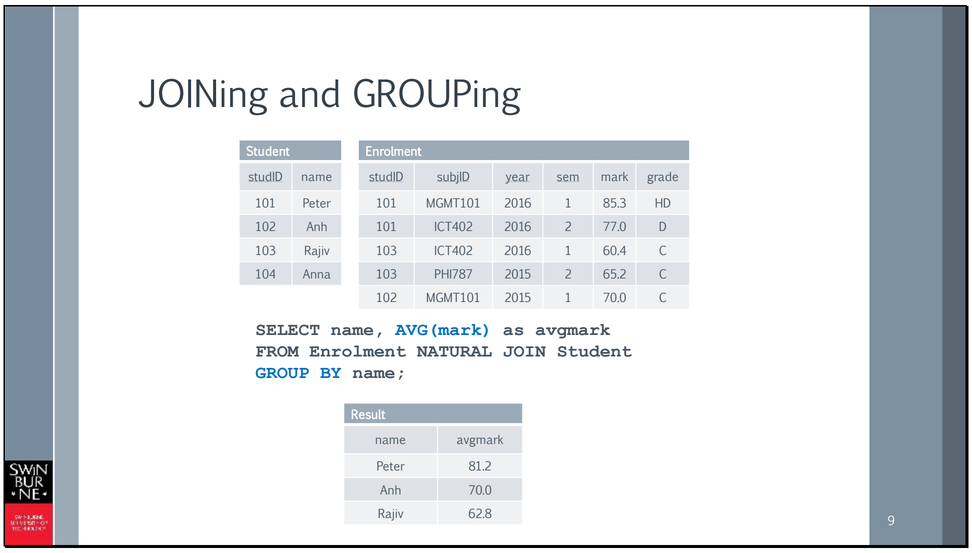
LEFT OUTER JOIN

RIGHT OUTER JOIN

8

There are inner and outer joins. What we have just seen were inner joins. The INNER keyword need not be used. JOINs are automatically inner joins when you specify nothing. Outer joins are all about what happens when rows don't have matches in the other table.
With left and right outer joins it matters which table is mentioned first, because with left outer joins we only include unmatched rows when they are in the table that's mentioned first. If you say neither left nor right, the default is full – meaning both tables have their unmatched rows included.

Slide 9



JOINing and GROUPing

Student

studID	name
101	Peter
102	Anh
103	Rajiv
104	Anna

Enrolment

studID	subjID	year	sem	mark	grade
101	MGMT101	2016	1	85.3	HD
101	ICT402	2016	2	77.0	D
103	ICT402	2016	1	60.4	C
103	PHI787	2015	2	65.2	C
102	MGMT101	2015	1	70.0	C

```
SELECT name, AVG(mark) as avgmark
FROM Enrolment NATURAL JOIN Student
GROUP BY name;
```

Result

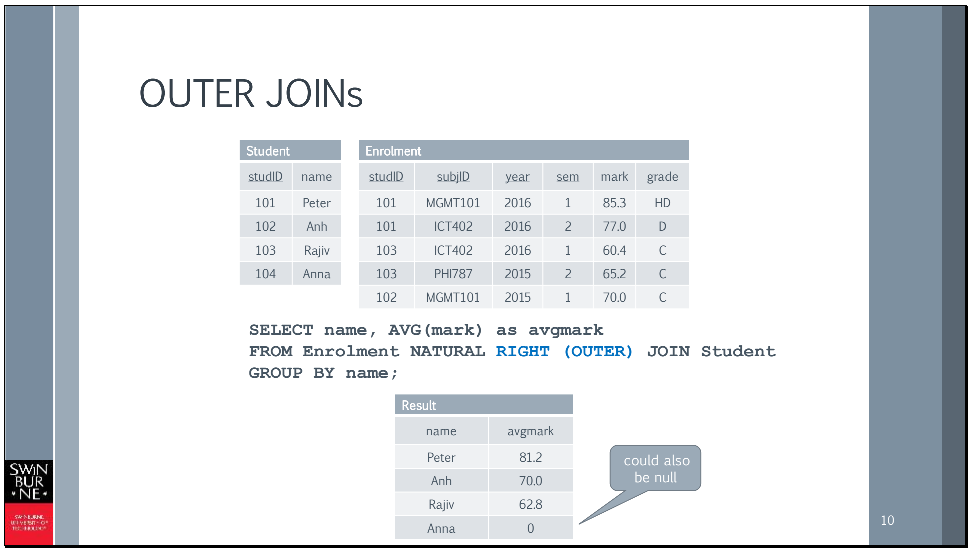
name	avgmark
Peter	81.2
Anh	70.0
Rajiv	62.8

Let’s look a more interesting example where we average students’ grades. Unlike in the previous query we won’t put in an restriction on the year. We want the average performance to date. Using the aggregate function average, we have to do a grouping. Since we want to have the performance per student, we group by the student.
You’ll notice that none of the columns in the Enrolment table is included. This makes sense, because we want the student’s performance overall, so we cannot specify semesters or years. We could, however, include the student id, but if we did, we would have to include it in the GROUP BY clause as well. Database systems don’t take kindly to including columns that are neither aggregates nor mentioned in the GROUP BY clause.

Having said that, in the current query we are relying on the fact that names are unique while in practice they are not. If you have two Peters or two Anhs among your students, all their marks get mixed up. For this to work, we are assuming that the names are unique.

Now we have a nice listing of all students’ names and average marks. But you’ll have noticed – Anna is missing.

Slide 10



OUTER JOINs

Student

studID	name
101	Peter
102	Anh
103	Rajiv
104	Anna

Enrolment

studID	subjID	year	sem	mark	grade
101	MGMT101	2016	1	85.3	HD
101	ICT402	2016	2	77.0	D
103	ICT402	2016	1	60.4	C
103	PHI787	2015	2	65.2	C
102	MGMT101	2015	1	70.0	C

```
SELECT name, AVG(mark) as avgmark
FROM Enrolment NATURAL RIGHT (OUTER) JOIN Student
GROUP BY name;
```

Result

name	avgmark
Peter	81.2
Anh	70.0
Rajiv	62.8
Anna	0

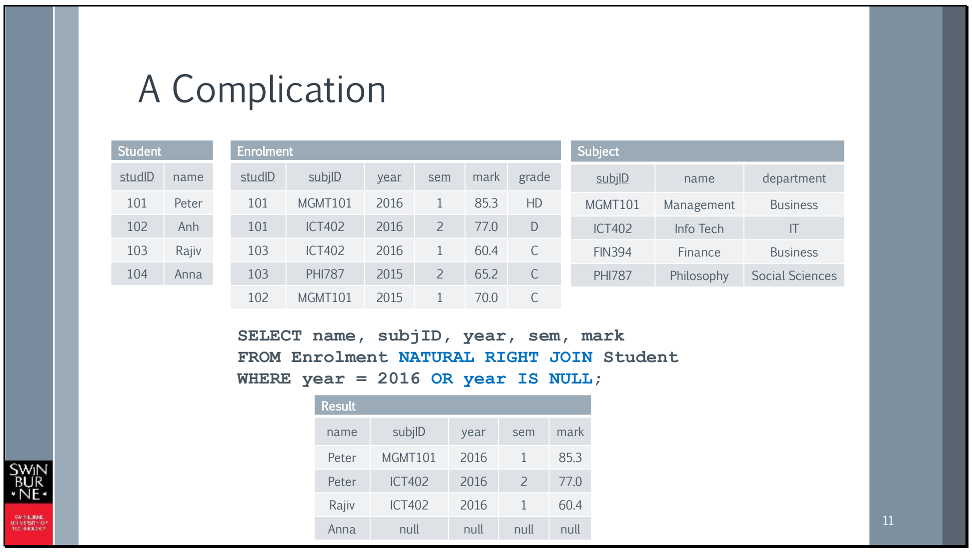
could also be null

10

Because Anna has no enrolment entries, this tuple in the student table has no matches in the enrolment table. This is why an inner join will not pick it up. If we specify a right join – it has to be outer because of the right keyword, so the outer keyword is optional – Anna is included. In MySQL, the value for average in MySQL is zero. This is an arbitrary choice. Not having entries for the mark column in enrolment is tantamount to having null entries. There are no entries, so mark is unknown. You'll find that some database products will opt for null, some for zero in such cases.
Note that we need a right join here because the student table is on the right hand side. A left join would include rows without matches from enrolment. But if we have implemented enrolment properly, there will be no enrolments without students, so left joins will have the same effect as inner joins – Anna will be missing.

Similarly, if we state outer join or full outer join, the result looks the same as it does here, because the additional row will be included, but there will be no additional rows from the enrolment table.

Slide 11



A Complication

Student	
studID	name
101	Peter
102	Anh
103	Rajiv
104	Anna

Enrolment					
studID	subjID	year	sem	mark	grade
101	MGMT101	2016	1	85.3	HD
101	ICT402	2016	2	77.0	D
103	ICT402	2016	1	60.4	C
103	PHI787	2015	2	65.2	C
102	MGMT101	2015	1	70.0	C

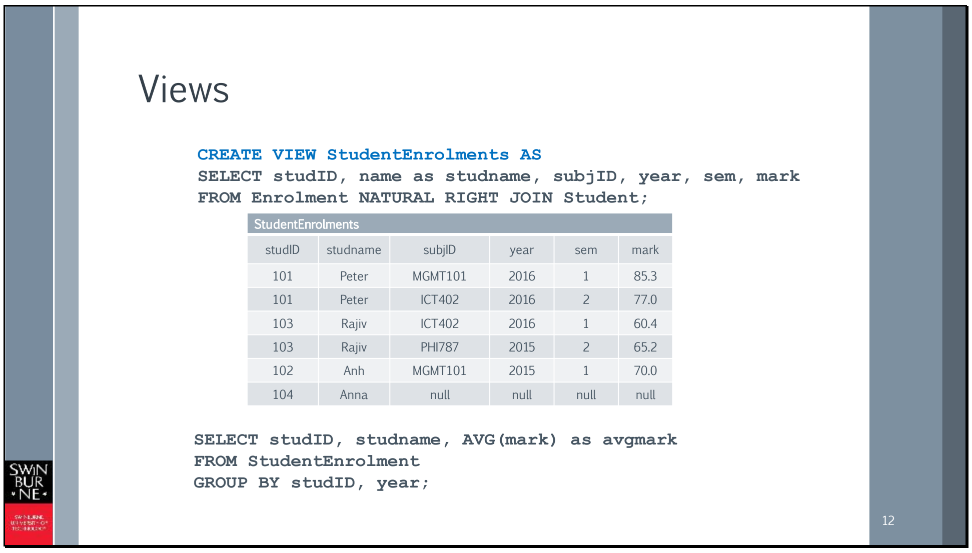
Subject		
subjID	name	department
MGMT101	Management	Business
ICT402	Info Tech	IT
FIN394	Finance	Business
PHI787	Philosophy	Social Sciences

```
SELECT name, subjID, year, sem, mark
FROM Enrolment NATURAL RIGHT JOIN Student
WHERE year = 2016 OR year IS NULL;
```

Result				
name	subjID	year	sem	mark
Peter	MGMT101	2016	1	85.3
Peter	ICT402	2016	2	77.0
Rajiv	ICT402	2016	1	60.4
Anna	null	null	null	null

In the first join query we looked at, we had a restriction on the year. And Anna was missing, because we had an inner join and Anna has no enrolments. But in this case, changing to an outer join won’t include Anna, because even when we include students without enrolments, the restriction on the year means that Anna will not be included. Why is this?
When a student has no enrolment but we include the student anyway using an outer join, all the fields from the table that has no entry for this student will be null.
Hopefully you remember that null values never evaluate to true or false, so we really can’t tell whether Anna’s nonexisting enrolment has a year of 2016.
Therefore you have to add the clause OR year IS NULL to make Anna appear in the result.
This is just in case you wondered. If you can’t get your head around this, it doesn’t matter. It will all become clear when you use SQL professionally.

Slide 12



Views

```
CREATE VIEW StudentEnrolments AS
SELECT studID, name as studname, subjID, year, sem, mark
FROM Enrolment NATURAL RIGHT JOIN Student;
```

StudentEnrolments

studID	studname	subjID	year	sem	mark
101	Peter	MGMT101	2016	1	85.3
101	Peter	ICT402	2016	2	77.0
103	Rajiv	ICT402	2016	1	60.4
103	Rajiv	PHI787	2015	2	65.2
102	Anh	MGMT101	2015	1	70.0
104	Anna	null	null	null	null

```
SELECT studID, studname, AVG(mark) as avgmark
FROM StudentEnrolment
GROUP BY studID, year;
```

12

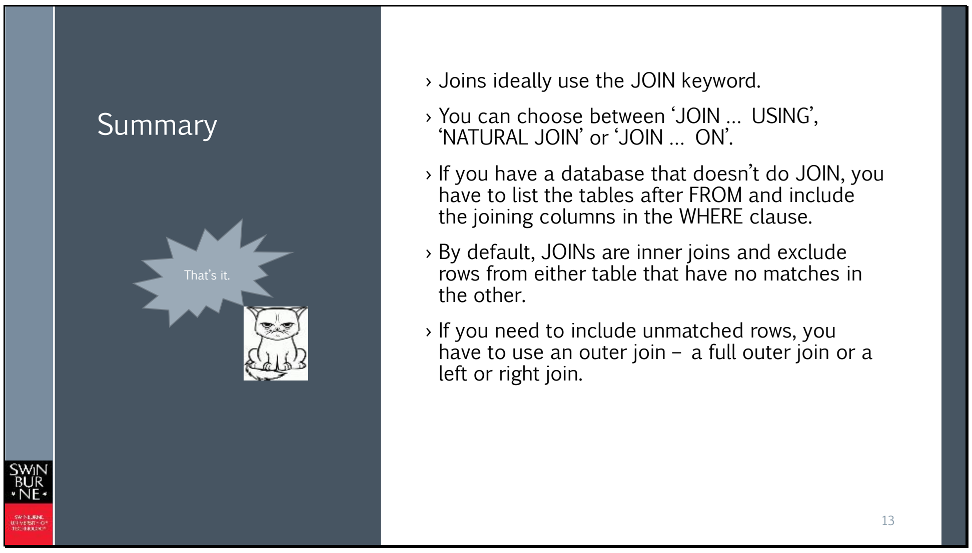
Views are simply stored queries. One of the advantages is that stored queries are pre-optimised and execute faster.
Another is that you can regulate access to date using views – you can prohibit a user's access to the base table and give him access to a view instead. Views can omit attributes as well as tuples, so you can limit access both horizontally and vertically.
Amazingly, views can actually be updated if they are constructed in a certain way. The result of an update of a view makes changes to the tables involved in a view.

Another advantage is that views make queries easier. Many queries get very complex and when users struggle to understand them, mistakes are made. Querying a view means the SELECT statement that queries the view can be simpler than the statement needed to get the same information from the tables.

The example demonstrates this. First we create a view StudentEnrolments. It joins the student and enrolment tables to include the student name. So when we investigate a student's performance per year, we no longer have to include the JOIN in the query.

This should not give you the impression that you can't use aggregates in VIEWs. You can. Essentially, anything that works as a SELECT statement – with a few exceptions – can also be created as a VIEW.

Slide 13



Summary

- Joins ideally use the JOIN keyword.
- You can choose between 'JOIN … USING', 'NATURAL JOIN' or 'JOIN … ON'.
- If you have a database that doesn't do JOIN, you have to list the tables after FROM and include the joining columns in the WHERE clause.
- By default, JOINs are inner joins and exclude rows from either table that have no matches in the other.
- If you need to include unmatched rows, you have to use an outer join – a full outer join or a left or right join.

Here is the summary of the presentation. Have a read before starting the quiz.