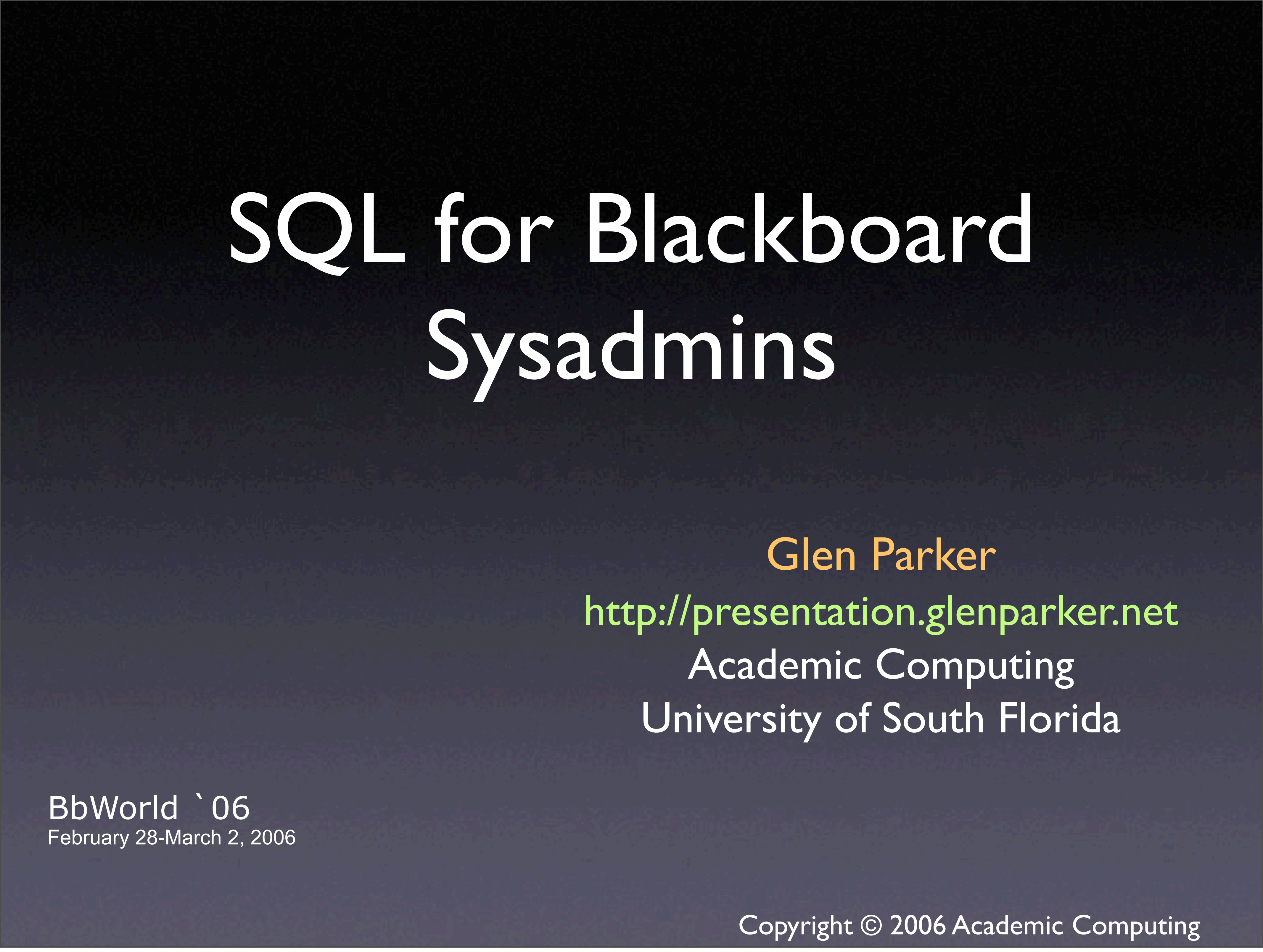


SQL for Blackboard Sysadmins

Glen Parker
http://presentation.glenparker.net
Academic Computing
University of South Florida

BbWorld `06
February 28-March 2, 2006

Good Morning
This SQL for Blackboard admins.
A quick bit of administration before we get started.
These slides, along with more complete code samples, are available on my website at the address on the screen.

Enough with the administration, we've a lot of SQL to cover, and it's only 7am.

First, let me answer the question "Why SQL"? Why should you spend the next 50 minutes with me and not next door learning how to "Transform the Student Experience" with Kettering University.

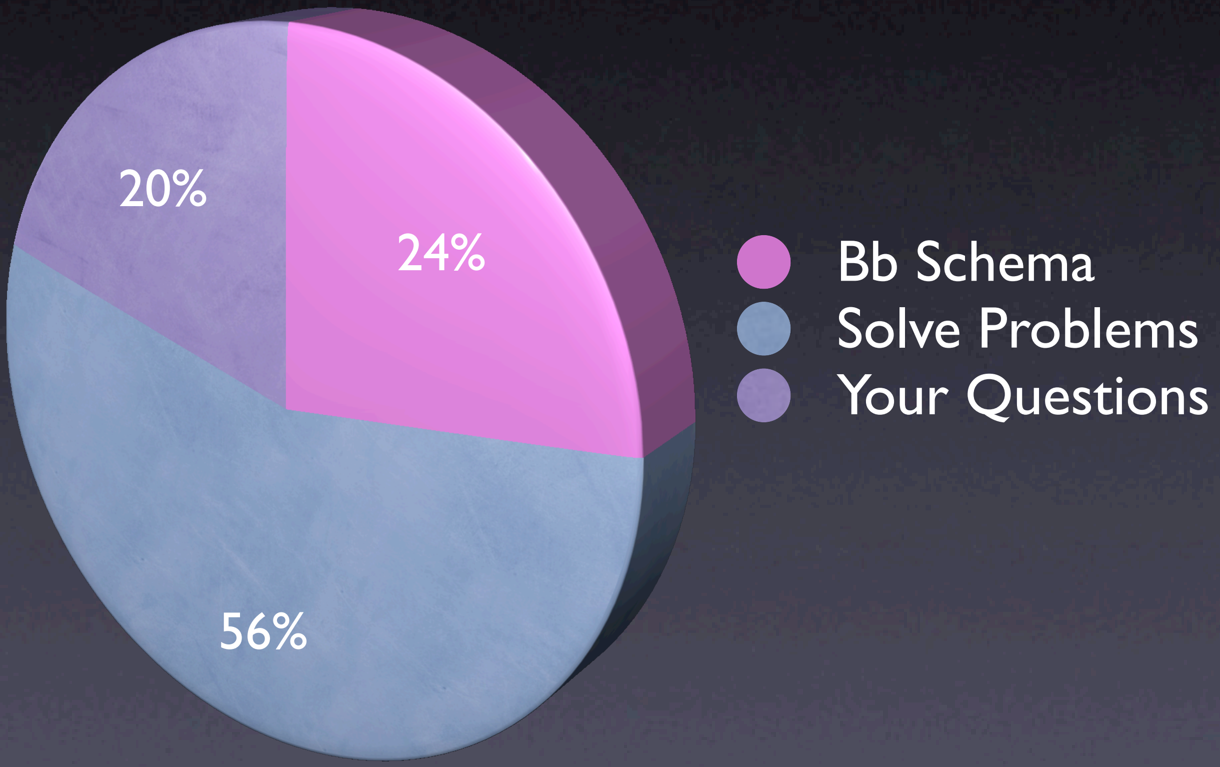
I'll give you three reasons.

First, Blackboard can't answer every question you want to ask of it. There's simply too many possible questions you can ask for Blackboard to provide a web interface for all of them. But with access to the data in your database, you can ask all the question you want. No need to wait for Blackboard 9.2 for your answer.

Second, it's possibly faster to ask your question directly of the database rather than go through the web interface. Think about this. Each web page you load on Blackboard requires a web server connection, a tomcat server thread, a database connection, and network resources to move all your data around. By talking directly to the database you can avoid all that and save those resources for your customers.

Third, writing SQL queries is fun. Especially at 7:03 in the morning. SQL is very different from most other 3rd generation languages, and writing queries tends to be more like solving the morning sudoku puzzle.

Today



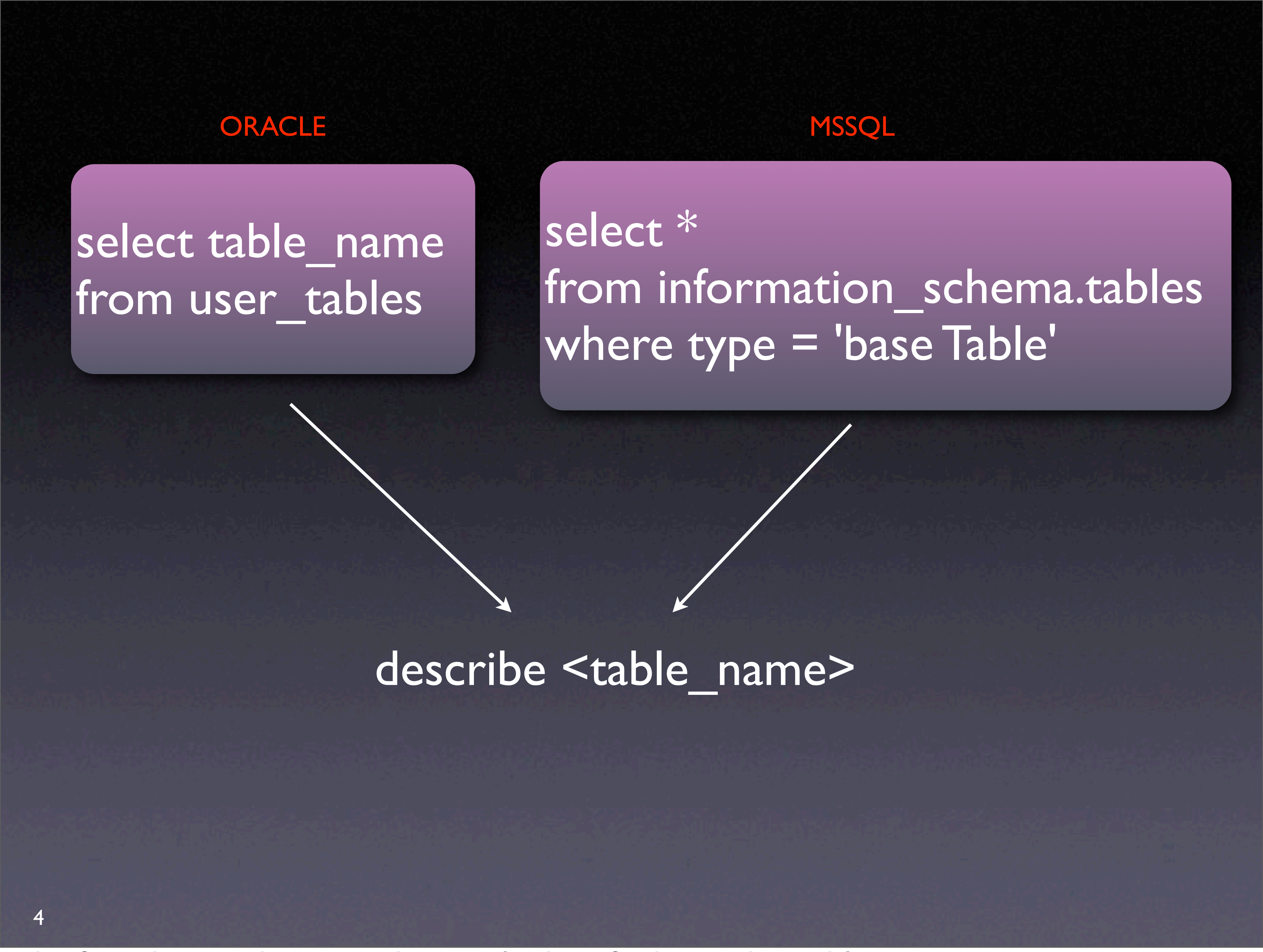
What are we going to explore today.
First I'll spend some time exploring the Blackboard schema. This will be the poor persons introduction to database design, Blackboard style.

Then we'll spend the lions share of time resolving problems that I've already solved.

Then We'll wrap it up looking by beyond the SQL console, and open the floor to try and solve your questions.

What Does A Blackboard Table Look Like ?

What does a table look like. I’ve dissected one for you, here
If you never looked at the Bb database, this will be new and interesting.



ORACLE

```
select table_name
from user_tables
```

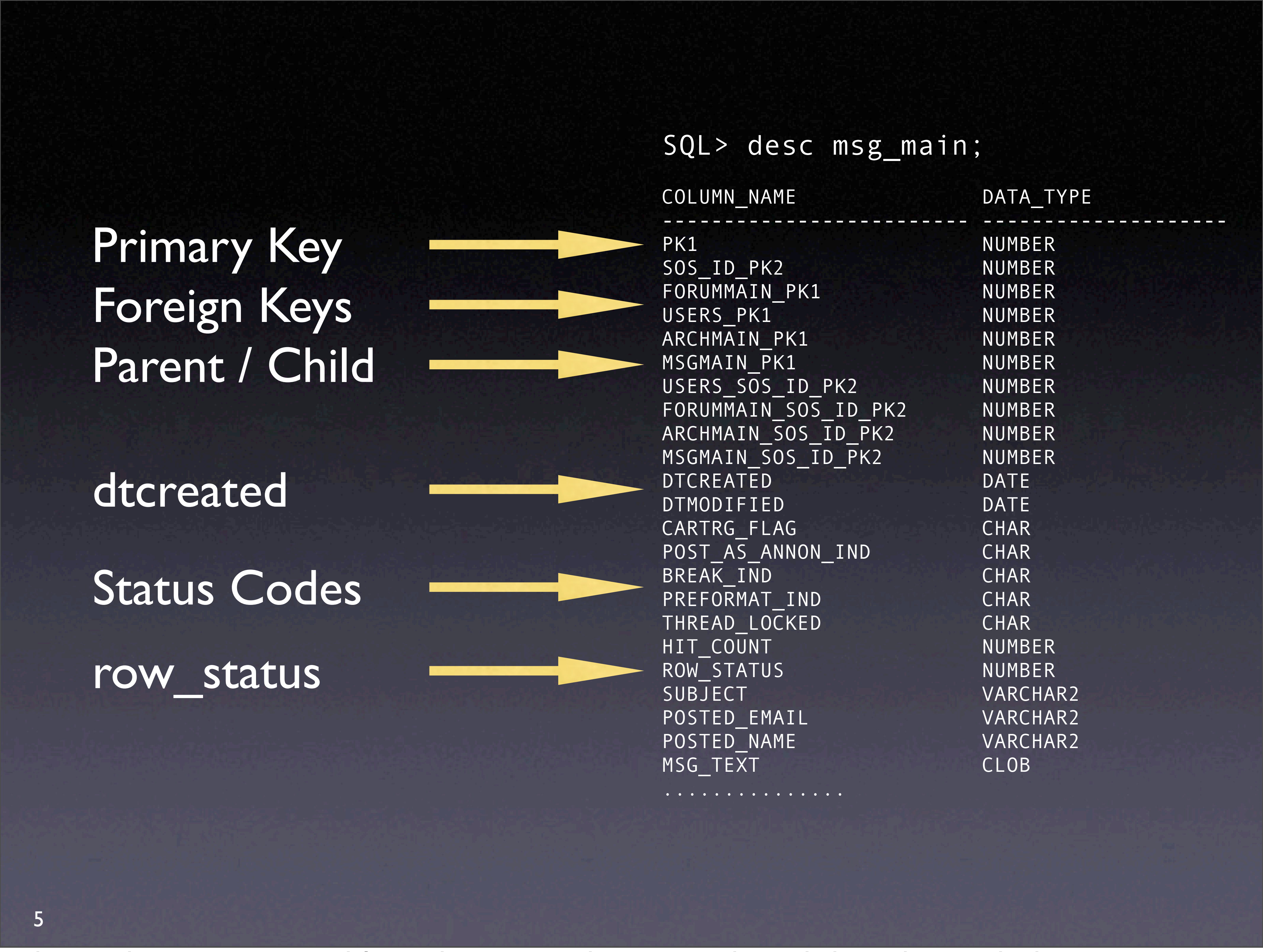
MSSQL

```
select *
from information_schema.tables
where type = 'base Table'
```

```
describe <table_name>
```

The first thing to know is where to look to find out what tables are on your system. These two queries will give you that
oracle run this command
– – – mssql run the other command

Once you've identified a table that has an interesting name, you can find out what columns are in that table with the describe command. simply describe and the name of the table. What you end up with is something that looks like this. . . . .



Primary Key

Foreign Keys

Parent / Child

dtcreated

Status Codes

row_status

```
SQL> desc msg_main;

COLUMN_NAME                DATA_TYPE
-------------------------  --------------------
PK1                        NUMBER
SOS_ID_PK2                 NUMBER
FORUMMAIN_PK1              NUMBER
USERS_PK1                  NUMBER
ARCHMAIN_PK1               NUMBER
MSGMAIN_PK1                NUMBER
USERS_SOS_ID_PK2           NUMBER
FORUMMAIN_SOS_ID_PK2       NUMBER
ARCHMAIN_SOS_ID_PK2        NUMBER
MSGMAIN_SOS_ID_PK2         NUMBER
DTCREATED                  DATE
DTMODIFIED                 DATE
CARTRG_FLAG                CHAR
POST_AS_ANNON_IND          CHAR
BREAK_IND                  CHAR
PREFORMAT_IND              CHAR
THREAD_LOCKED              CHAR
HIT_COUNT                  NUMBER
ROW_STATUS                 NUMBER
SUBJECT                    VARCHAR2
POSTED_EMAIL               VARCHAR2
POSTED_NAME                VARCHAR2
MSG_TEXT                   CLOB
...............
```

This is the msg_main table and it stores discussion board threads. Each discussion board thread in any discussion board gets a row in this table.

Beyond storing discussion threads, this table is useful for spotlighting columns that are found in many Blackboard tables.

First, there's a primary key. Every table that Blackboard designed has a primary key labeled "pk1". These are used to join tables together

This table also has some foreign key references. In particular, this table references forummain which tells us which forum the thread belongs to; while the Users pk1 identifies the user who posted the thread.

This table also has a parent child key, which blackboard uses to stored nested objects. In this case, the msgmain pk1 tells us if our thread is a top level thread, or if it's in fact a reply to another thread. If msgmain pk1 is NULL, then it's a top level thread. In there's a value in msgmain pk1, then that value is the parent thread to our reply.

There's a dtcreated column, which blackboard uses to record when the row was inserted in the table. The dtmodified column tells us when it was last updated. These are really useful for telling instructors when data may have changed in their course.

Then there are status codes. These are used to customize the state of the row. For this table, we have the option of posting threads anonymously, and the option of posting threads with preformatted html. Figuring our the values that different status codes can take is strictly a matter of trial and error, on your testing servers.

Important safety tip: Never experiment on your production servers.

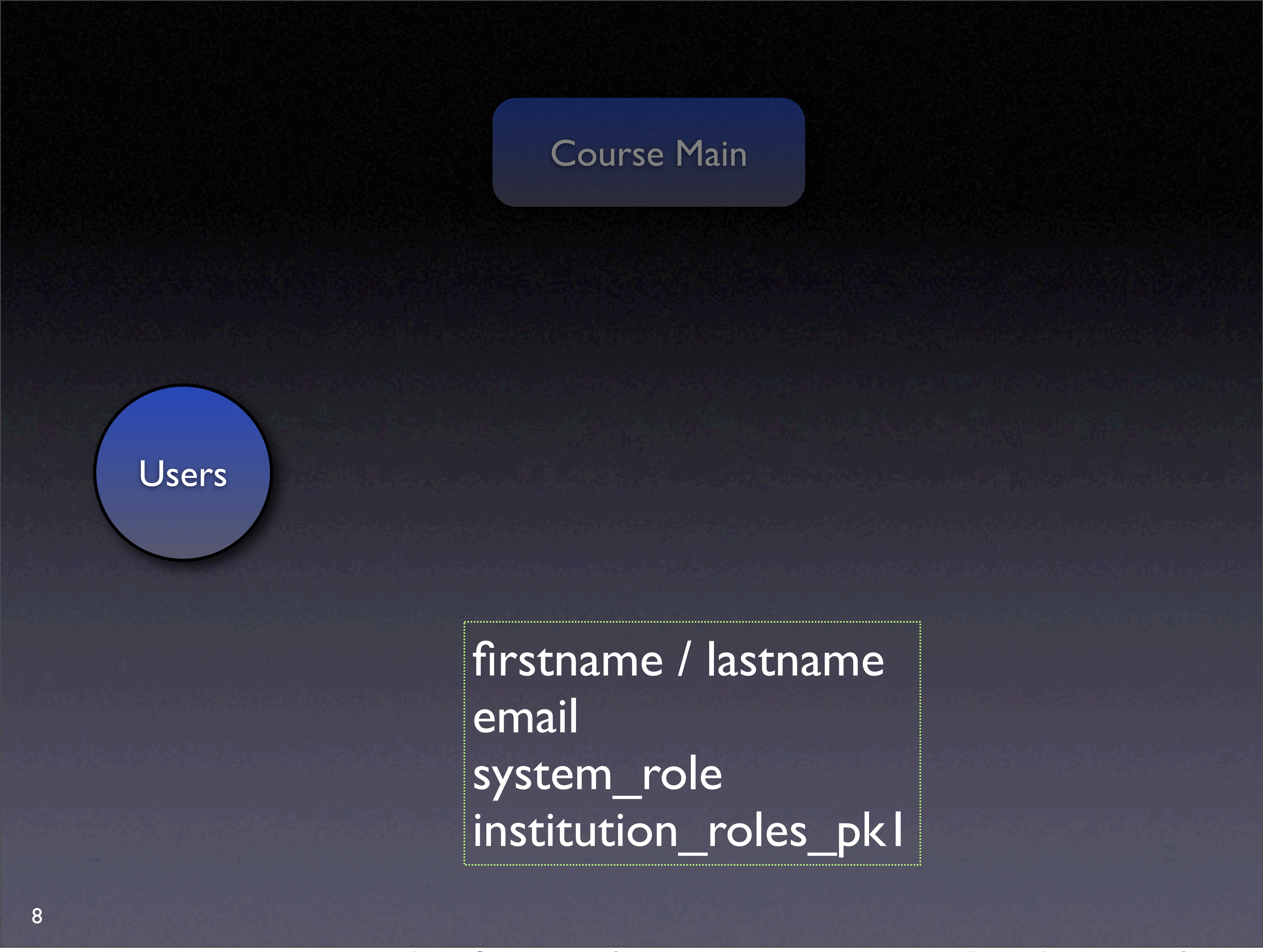
What Tables Should We Be Interested In ?

So that's one table down. At last count there were over 200 tables that Blackboard uses.
I've gone ahead and identified 8 other tables that I tend to use more often than the rest.
What follows will be a picture that I'll build up over the next few slides.
In that picture I'll try to describe the key attributes of each table, and show the relationship between them.

Course Main

course_id
course_name
duration / start_date / end_date
available_ind

The first table is course_main
Each row in course main identifies a single course site on your system
Table contains basic , top level course information like course_id, course_name, start and end date, duration,
- if the course was made unavailable by the instructor,
- if the course is in fact an organization, the data source of the course if snapshots are used.



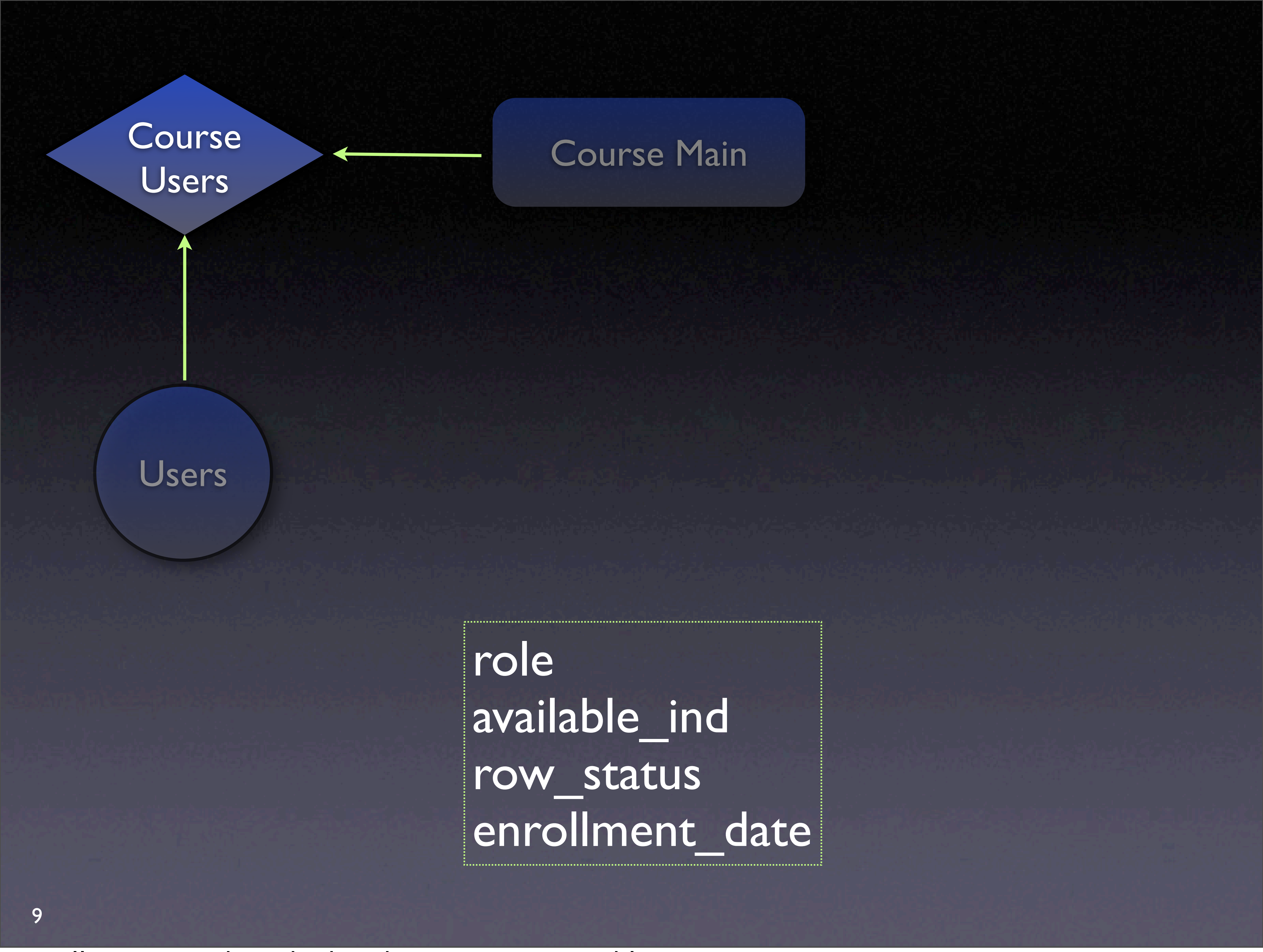
Course Main

Users

firstname / lastname
email
system_role
institution_roles_pk1

Just as row in course_main identifies a single course, a row in users describes a single user account. Here you'll find basic information about your users, name, email, gender, address, and any other information you load about your user population. System admin privileges are granted via this table .

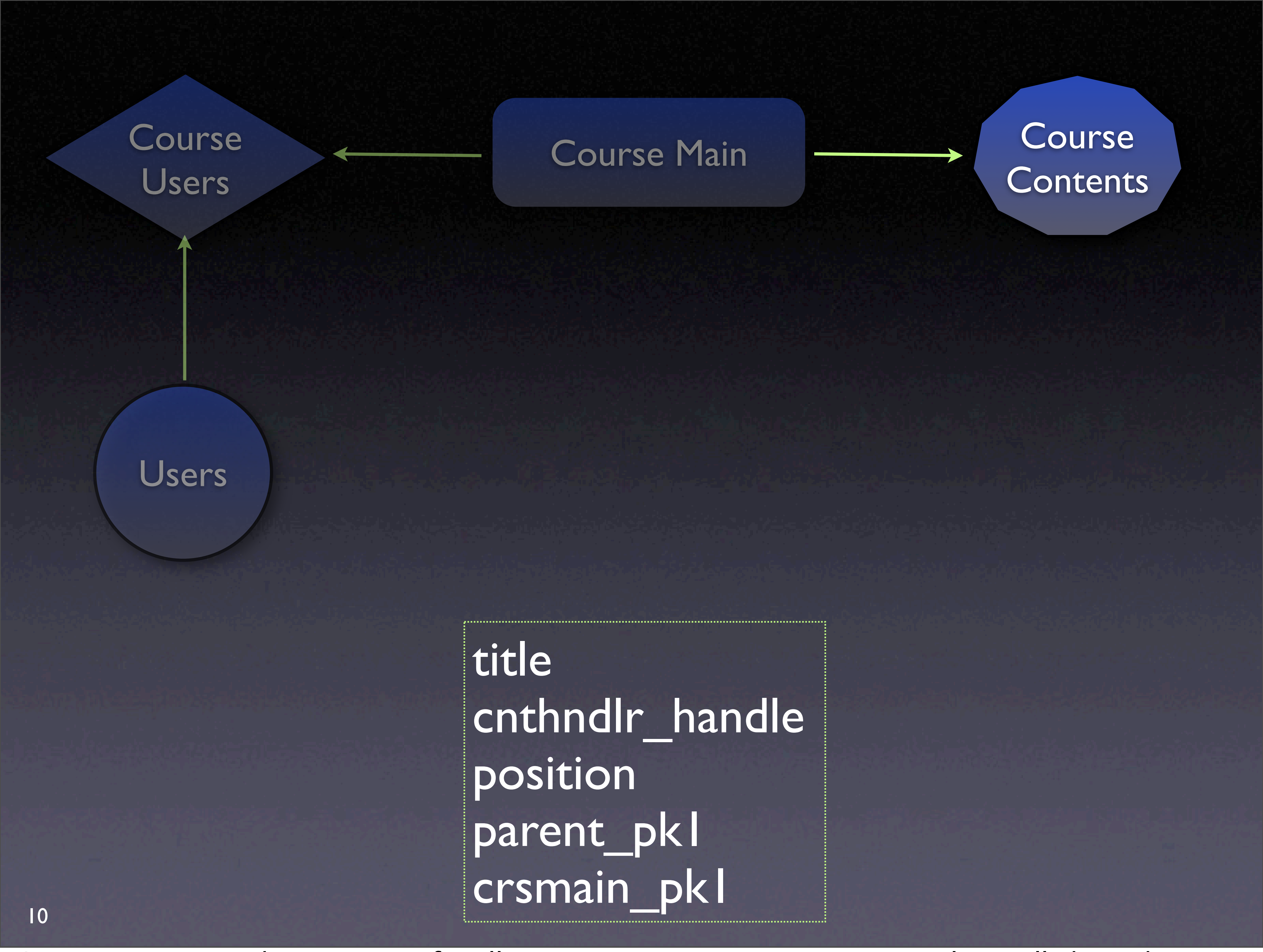
You can also find there primary institution role in this table. Secondary roles are stored in a different table.



role
available_ind
row_status
enrollment_date

enrollments are described in the course_users table.
One to one relation between course_main and users,
a row in course_users identifies a user enrolled in a course.
this table Includes role in the course(S,P,T), unavailable by instructor or disabled by snapshot
You can see when a person was added to the course by looking at the enrollment_date.

I’ve often use the state of all available and row_status to help determine why a student is not able to access a course. Many times the instructor has made the either the student or the course unavailable, many other times they were disabled by snapshot due to problems with their registration in Banner. You gotta know what’s wrong before you can fix it.

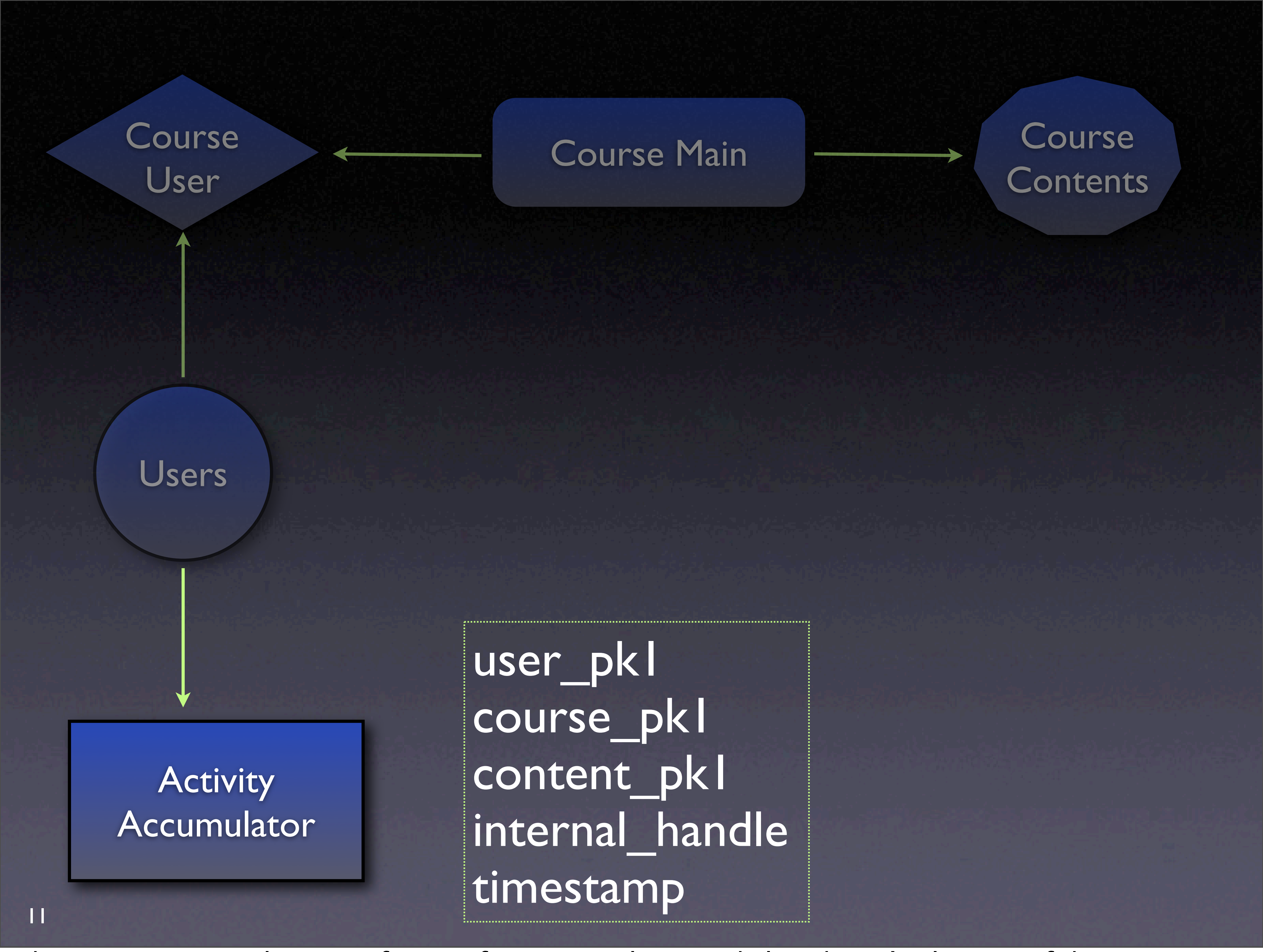


title
cnthndlr_handle
position
parent_pk1
crsmain_pk1

Course contents is the repository for all content items in a course. You can learn all about the types of content a course is using by looking here.

cnthndlr_handle shows the different types of content files, Within a Blackboard course, there are a number of content types available for the instructor to choose from. Plain files, folders, Assignments, URL's, Learning Units, so on.   Each content type available to instructors has a unique cnthndlr_handle value in the course_contents table. Looking for certain values of cnthndlr_handle can tell you if a particular type of content is being used by instructors.

position shows where on the page it will be listed,
parent_pk1 describes the folder that this content item belongs in. If this is a top level content item, then parent_pk1 is null.

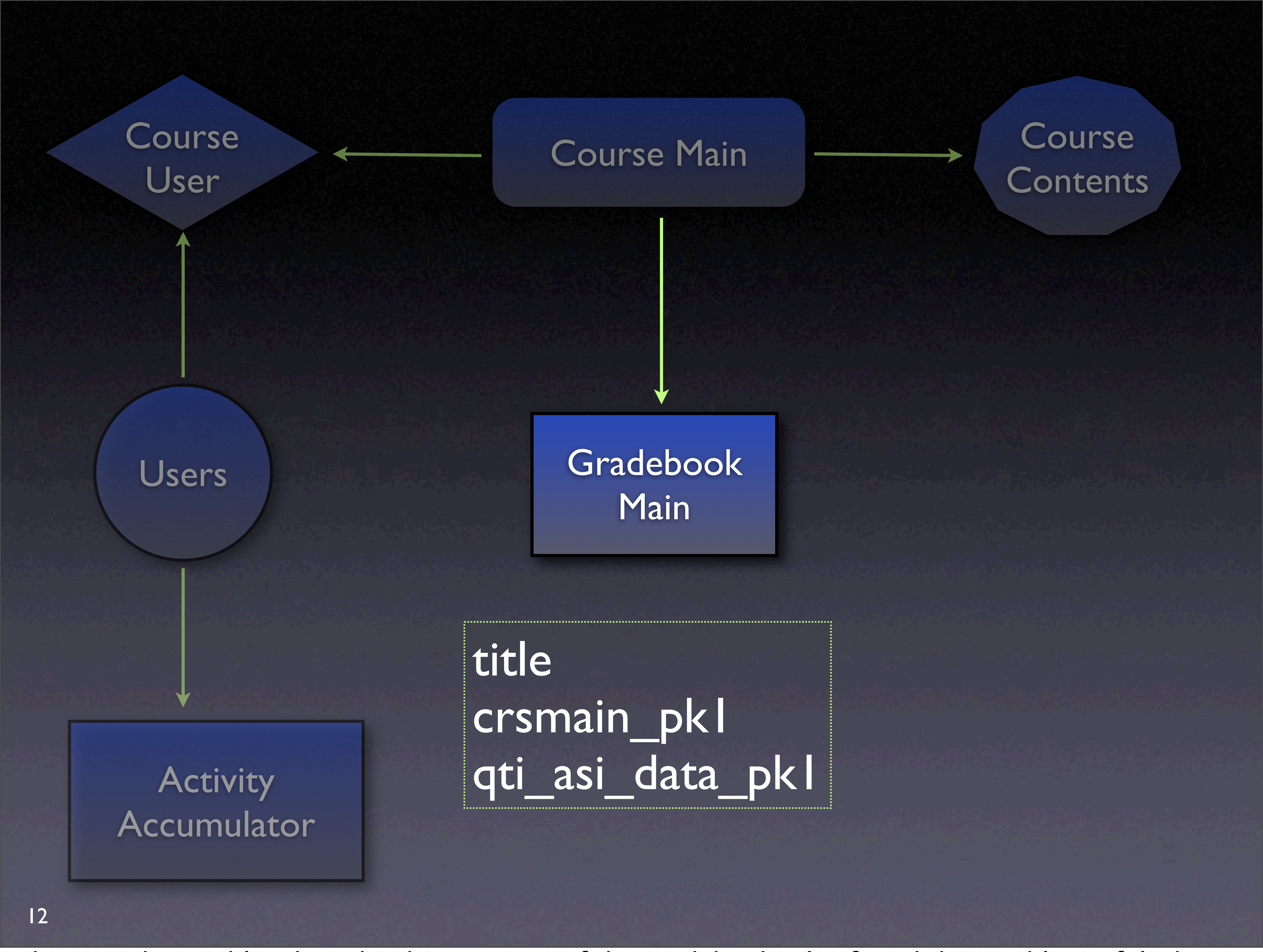


user_pk1
course_pk1
content_pk1
internal_handle
timestamp

The activity_accumulator is infamous for growing large and slow, but it's also one of the most interesting tables in your system. Almost every page a user visits in your system gets recorded here. It's like your apache access logs but with better user tracking. I've used this table many times to prove to instructors who was responsible for making certain undesirable changes to their course.

If the user has logged in, then there activity is tied to their user_pk1. If they are moving in a course, the course_pk1 is also set.
internal_handle described which of over 230 different page types available to be tracked.
content_pk1 tells us if they accessed a particular piece of content.
timestamp tells us when it all went down.

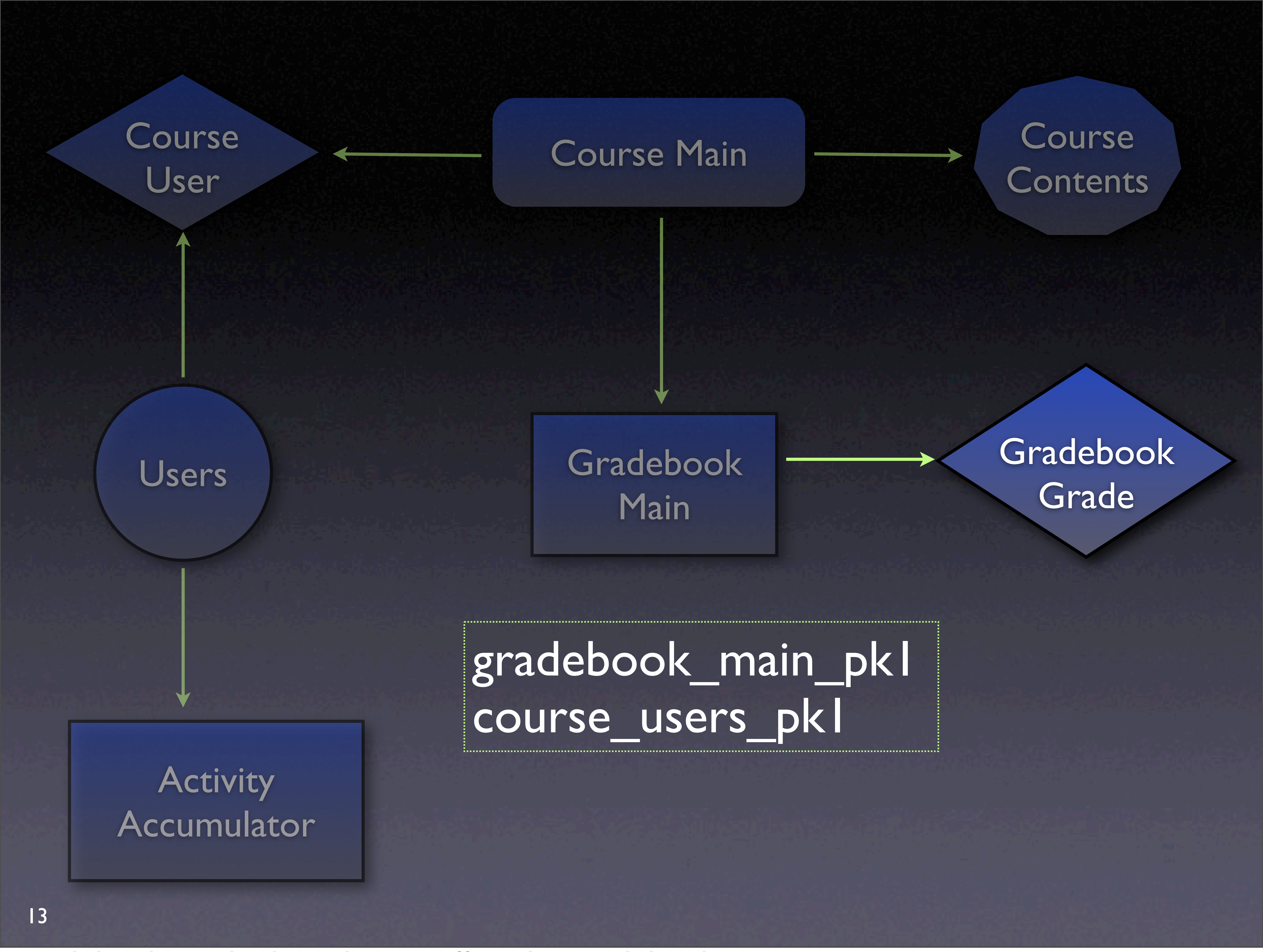
Blackboard even went so far as to index this table very well, making it possible to run many different reports in a reasonable time.



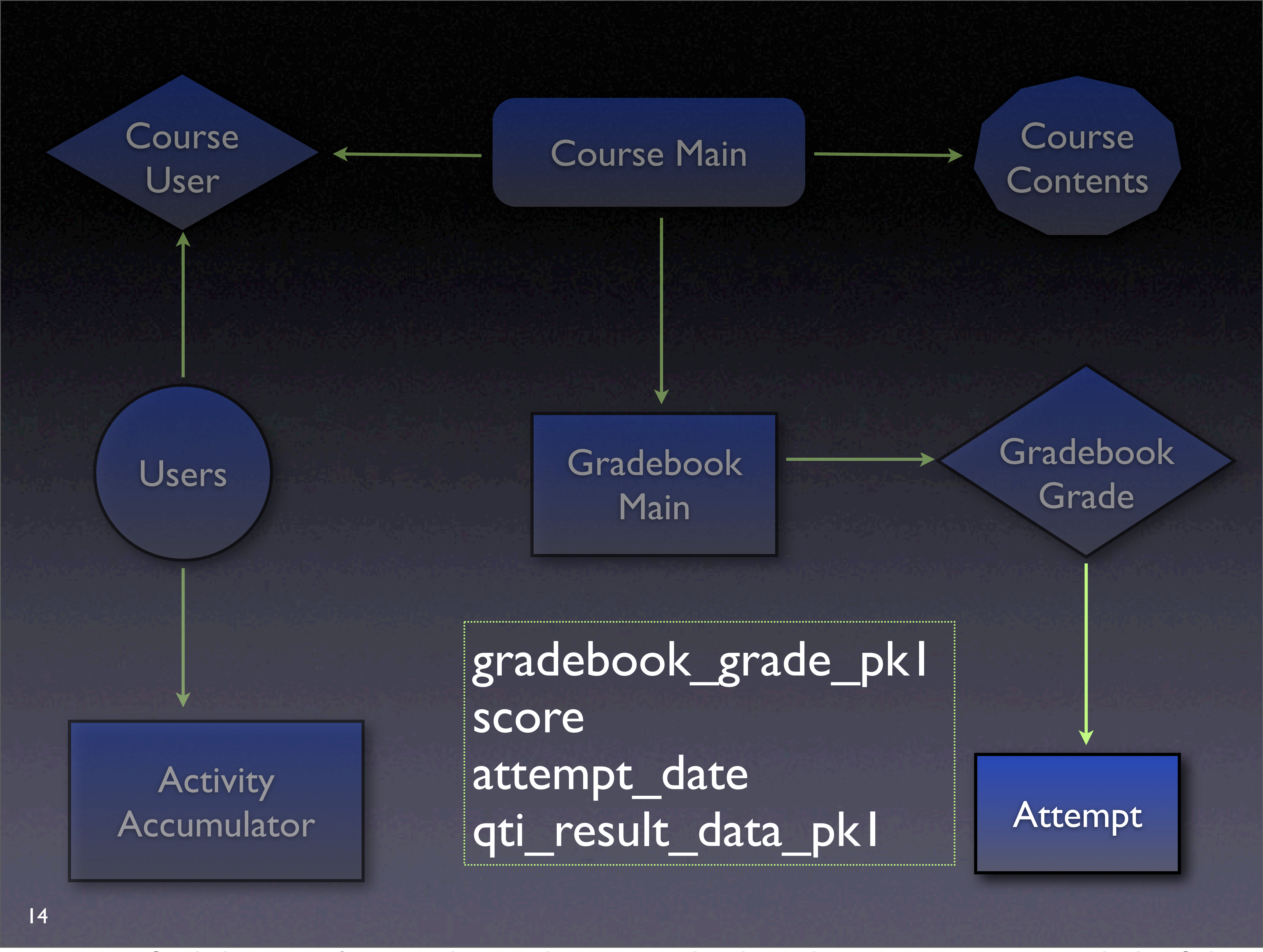
title
crsmain_pk1
qti_asi_data_pk1

The next three tables describe the structure of the gradebook. I've found these table useful when tracking down discrepencies in an instructors gradebook. I've also used these table to build a primitive early warning system for instructors. This tool tells when students are falling behind in their homework submissions, when it's been so many weeks since they last submitted anything.

Gradebook Main describes a column in the gradebook. Every gradebook column gets a row in this table. If the gradebook column happened to be created as part of a test or survey, then the column also points to the qti_asi_data xml blob that generated it. the qti_asi_data blob contains the questions that make up the quiz or survey.



gradebook grade describes a cell in the gradebook.
Relational table between course users and gradebook_main
It’s a holding place for a students attempt on a particular gradebook item.
Actual scores are not stored in this table, but rather in the attempt table.



gradebook_grade_pk1
score
attempt_date
qti_result_data_pk1

Here you find the actual score the student earned, when the attemp was recorded. If the gradebook item was a test or survey, then the attempt records points to the qti_result_data xml clob that contains the students answers.

Any Questions?
((PAUSE FOR QUESTIONS))
((Check Time))

Problems In Need of a SQL Solution

Knowing what the Blackboard schema looks like is like owning a bucket of hammers. Every problems that comes our way will start to look like a SQL coated 10 penny nail.

I have 6 such problems here that I've solved in the course of my day job. All the solutions involved SQL in some fashion.

I've picked these problems because they stand out in my mind as being interesting problems with even more interesting solutions.

Hopefully you'll agree.

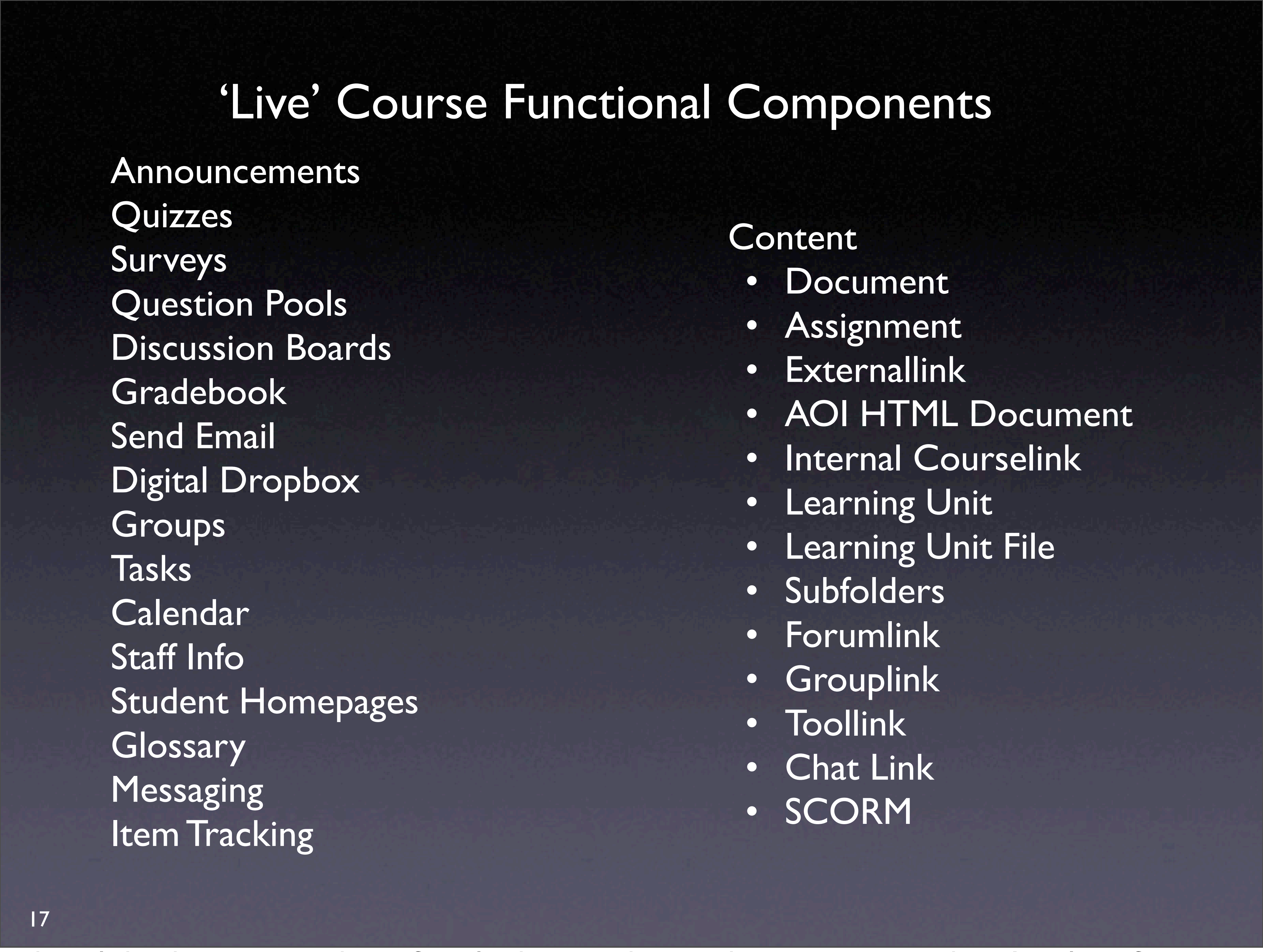
Course Use

One good problem is identifying how a Blackboard course is being used.
You want to know what blackboard tools your are faculty using in their courses.

I started this project on a lark a few years ago to see if I could tell which instructors were using the digital dropbox. Once I realized that I had full access to the information stored in the database, I started looking to identify other areas of courses that were being used by faculty. Today, It's grown into a project that has guaranteed me years of job security.

Here's an example of one of the facts I've been able to glean.
70% Bb instructors use the gradebook.

Why do we care?
We are working on another project to allow faculty to submit Grades from their Gradebook directly to Banner. Being able to prove that we had a well established user base played a large part in greenlighting our project over two other commercial alternatives.



'Live' Course Functional Components

Announcements
Quizzes
Surveys
Question Pools
Discussion Boards
Gradebook
Send Email
Digital Dropbox
Groups
Tasks
Calendar
Staff Info
Student Homepages
Glossary
Messaging
Item Tracking

Content
- Document
- Assignment
- Externallink
- AOI HTML Document
- Internal Courselink
- Learning Unit
- Learning Unit File
- Subfolders
- Forumlink
- Grouplink
- Toollink
- Chat Link
- SCORM

This slide shows a number of tools that can be used in a course.   There's a lot of pieces you could be interested in. The best part is, if Blackboard stored the data in the database then you can use SQL to look at it. You'll probably want to pick a few items from this list and try to learn for yourself where and how Blackboard stores the data for that item.

Remember, all of this is undocumented, so trial and error is the order of the day.

That and do all your trialing and erroring on a test server, never on production.
Two warnings on one day, and it's only 7:20 in the morning.

So Let's learn how to identify 4 of these.

Gradebook

```
select crsmain_pk1
from gradebook_main
group by crsmain_pk1
having count(*) > 2;
```

First course tool is the gradebook. A simple test of gradebook usage is to count the number of rows in the gradebook_main table for each course. Every column in the gradebook of a course gets a row in the gradebook_main table.
This includes the Total and Weighted Total column.
Since every course has at least two rows, then it makes sense to look for courses that have more than two rows in the gradebook_main table. Those that do have more than 2 rows are the courses that have added columns to the gradebook.

This query will return the course_main primary key for courses that pass the test. In and of itself, this primary key is not too useful. However. . .

Gradebook

```
select cm.course_id
from course_main cm, (
  select crsmain_pk1
  from gradebook_main
  group by crsmain_pk1
  having count(*) > 2
) inner
where inner.crsmain_pk1=cm.pk1
;
```

We can make the output prettier looking and more useful. Make our query a subquery, which we'll call 'inner'. We can then join the set of course main pk1's we get from inner to the set of course main pk1's we get from course_main itself, and select the course_id from course_main. It's a clean way to translate the crsmain-pk1's in gradebook_main to course_id's in course_main.

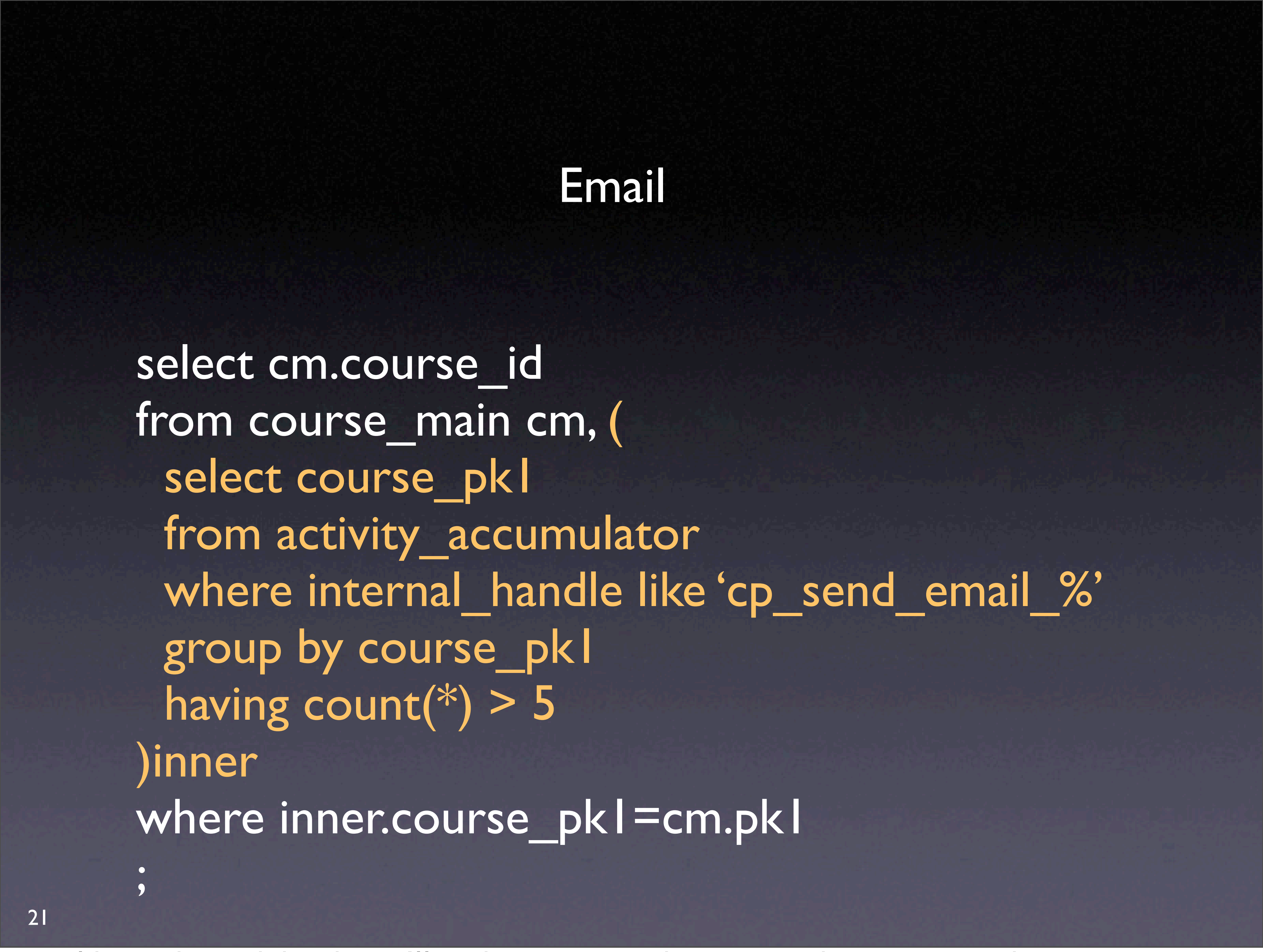
Email

```
select course_pk1
from activity_accumulator
where internal_handle like 'cp_send_email_%'
group by course_pk1
having count(*) > 5
```

Next we'll look for email. Blackboard doesn't store email messages in the database, so we'll have to make a best guess.
The activity accumulator records whenever an instructor clicks on one of the control panel's send email pages. That is, in order to actually send an email message from the control panel, they have to click two or three links to get to the email form. It's these clicks that are recorded in the activity_accumulator.

For purpose of this statistic, I've decided that if more than 5 hits are recorded, that email was sent. Why 5, that allows for two complete trips to the send email form plus an extra click.   My pathos says that 5 is a good estimate, my logos can't come up with a good counter arguement. my ethos doesn't care either way.

This query returns all course_pk1's that qualify under our send email assumption test.



Email

```
select cm.course_id
from course_main cm, (
  select course_pk1
  from activity_accumulator
  where internal_handle like ‘cp_send_email_%’
  group by course_pk1
  having count(*) > 5
)inner
where inner.course_pk1=cm.pk1
;
```

Just like with gradebook, we’ll make it prettier by joining the output to the course_main table. This is just like the gradebook, so we’ll move on.

Dropbox

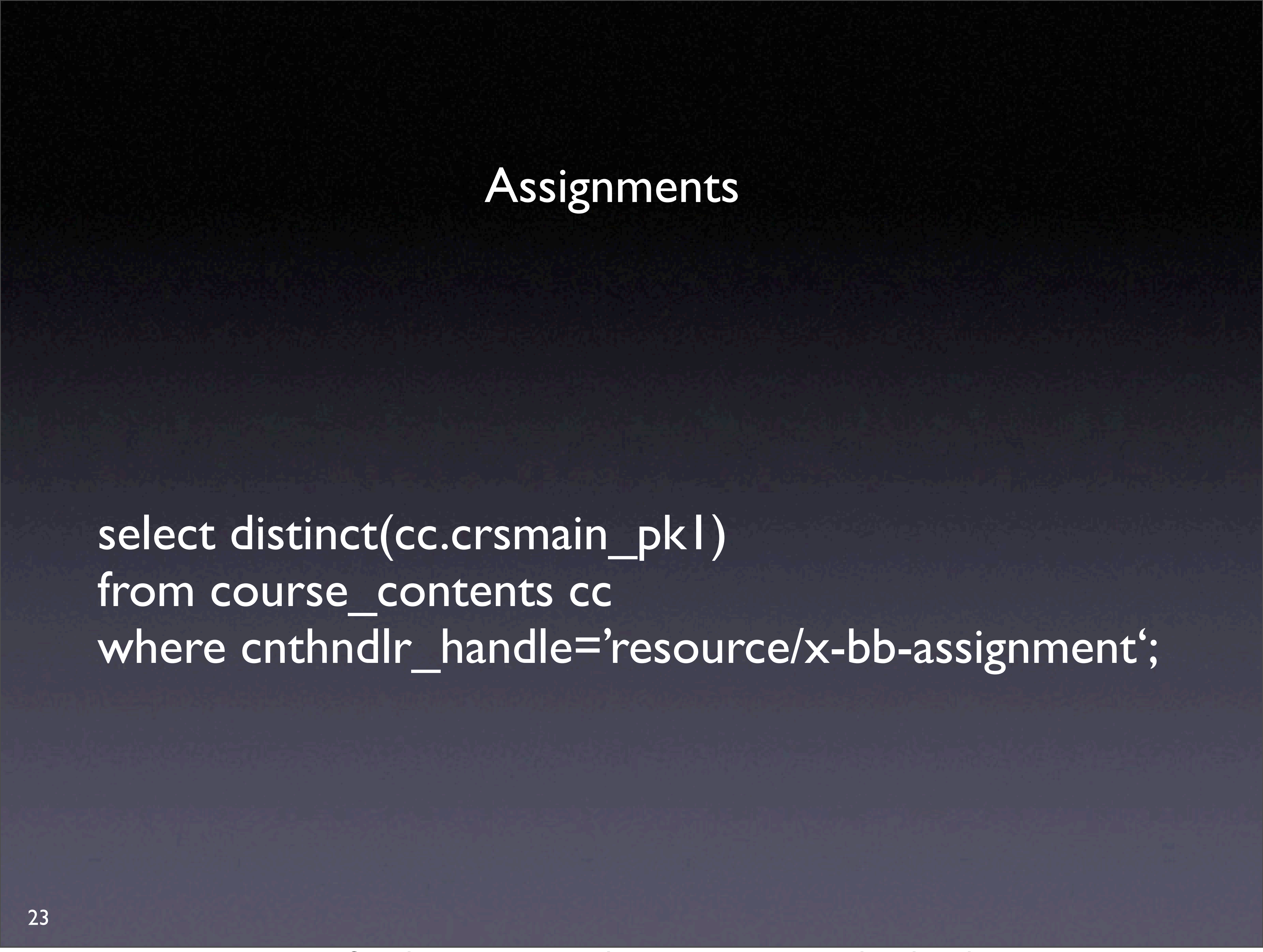
```
select distinct(u.email)
from course_user_uploads cuu
	,course_users cu
	,users u
where cuu.course_users_pk1=cu.pk1
and cu.users_pk1=u.pk1
and cu.role='P';
```

The dropbox is a hot topic, especially now that it's been deprecated. A lot of schools, ours included , want to know which faculty are still using the dropbox so we can let them know about alternatives, such as Assignments and SafeAssignment. This query tells us that.

Information about dropbox files are stored in the course_user_uploads table. This tables has information about the file uplaoded, and the course_users_pk1 of the person who uploaded it.

We need to join course_user_uploads table to course_users to determine the user and course that the upload belongs to. And since we're already joining tables, we'll join to the users table to get the email address of the instructors of the courses.

Finally, we use distinct to remove duplicate email addresses from the results.
What we have here are all instructor email addresses that have at least one file in the dropbox.



Assignments

```
select distinct(cc.crsmain_pk1)
from course_contents cc
where cnthndlr_handle=’resource/x-bb-assignment‘;
```

Last course component for this section is the assignments. the dropbox vs. assignments question is a common one. And it turns out to be a pretty easy question to answer. In the course_contents table, the cnthndlr_handle columns tells us about all file types you can select from the pull down menu when adding content to courses. Identify the handle for assignments, then select all courses that have one or more content items like that.

If a course has more than one assignment, they have more than one row in course_contents. We’ll use distinct again to remove the duplicate.

Assignments

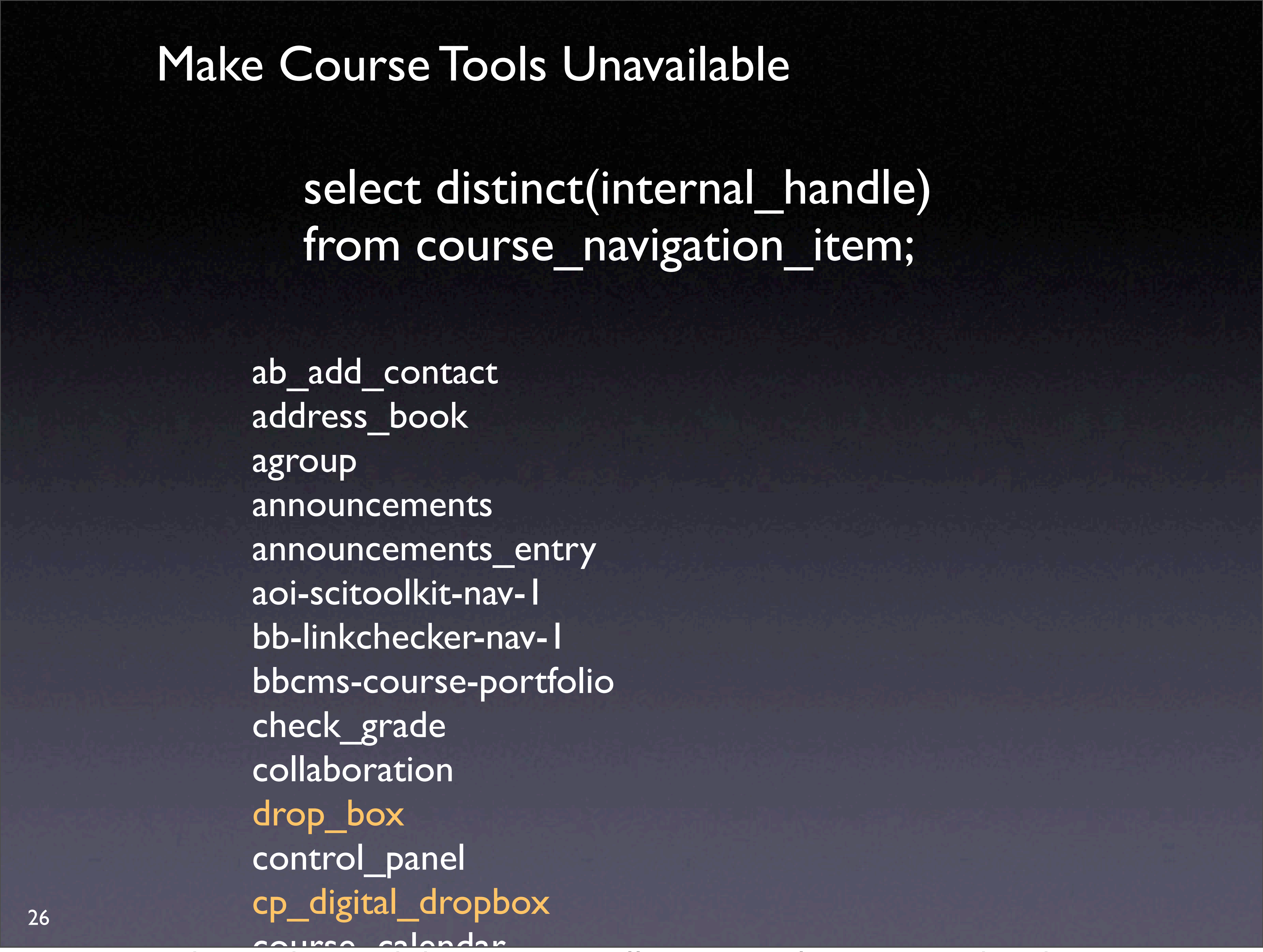
```
select cm.course_id
from course_main cm, (
    select distinct(cc.crsmain_pk1)
    from course_contents cc
    where cnthndlr_handle='resource/x-bb-assignment'
) inner
where inner.crsmain_pk1=cm.pk1
;
```

And again, we can make it pretty by joining with course_main.

Now that you've seen how to get information on some components, you'll be happy to know that most of the course components on that big list are found the same way. Search the BBADMIN mailing list archives for an Uber query I wrote a while ago that returns information on every component on that list, all in a single query. It's awesome, just ask my friend Jim from Alaska. Or you can visit my website for some Perl code that does the same thing.

Make Course Tools Unavailable

Another problem to solve.   At South Florida, we’ve made a decision to try to encourage faculty to switch from the slow, unreliable digital dropbox to any of the other options available to them.
But we don’t want to cut off access altogether, at least not yet. What we’d wanted is for the tool to be unavailable by default, so that most instructors won’t ever see tool in their course unless they deliberatly went to make it available.
Here’s how we made that happen.



Make Course Tools Unavailable

```
select distinct(internal_handle)
from course_navigation_item;
```

```
ab_add_contact
address_book
agroup
announcements
announcements_entry
aoi-scitoolkit-nav-1
bb-linkchecker-nav-1
bbcms-course-portfolio
check_grade
collaboration
drop_box
control_panel
cp_digital_dropbox
```

First we need to understand that access to all course tools is managed in the course_navigation_item table. Each course has a list of course tools and the status of that tool. The tools are identified by the internal_handle column.

So we need to identify the internal handle of the tool we which to disable. Run this query and you'll get a long list of a couple hundred possible handles. It'll look like this.

Now there may be more than one handle we're interested in. This happens when the tool is accessed in more than one place in a course. The drop box has entries in the students course tools area, and in the control panel for instructors. This query will list all possible handles. Trial and error will be needed to figure out which ones to make unavailable yada yada yada.

For our dropbox project, we'll need the handles "dropbox" and "cp digital dropbox"

Make Course Tools Unavailable

```
update course_navigation_item
set enabled_ind='N'
where internal_handle='drop_box'
or internal_handle='cp_digital_dropbox';
```

Now that we have the handle to disable, we can update course_navigation_item. Setting enabled_ind to N disabled the ability to click the link. If this is a course tool them the link simple disappears. If it's a control panel tool, then the control panel link is greyed out. The instructor can go to the appropriate place in either manage tool or manage menu items to reenable access to the tool.

Make Course Tools Unavailable

```
update course_navigation_item
set enabled_ind='N'
where (internal_handle='drop_box'
or internal_handle='cp_digital_dropbox')
and crsmain_pk1 = ?
```

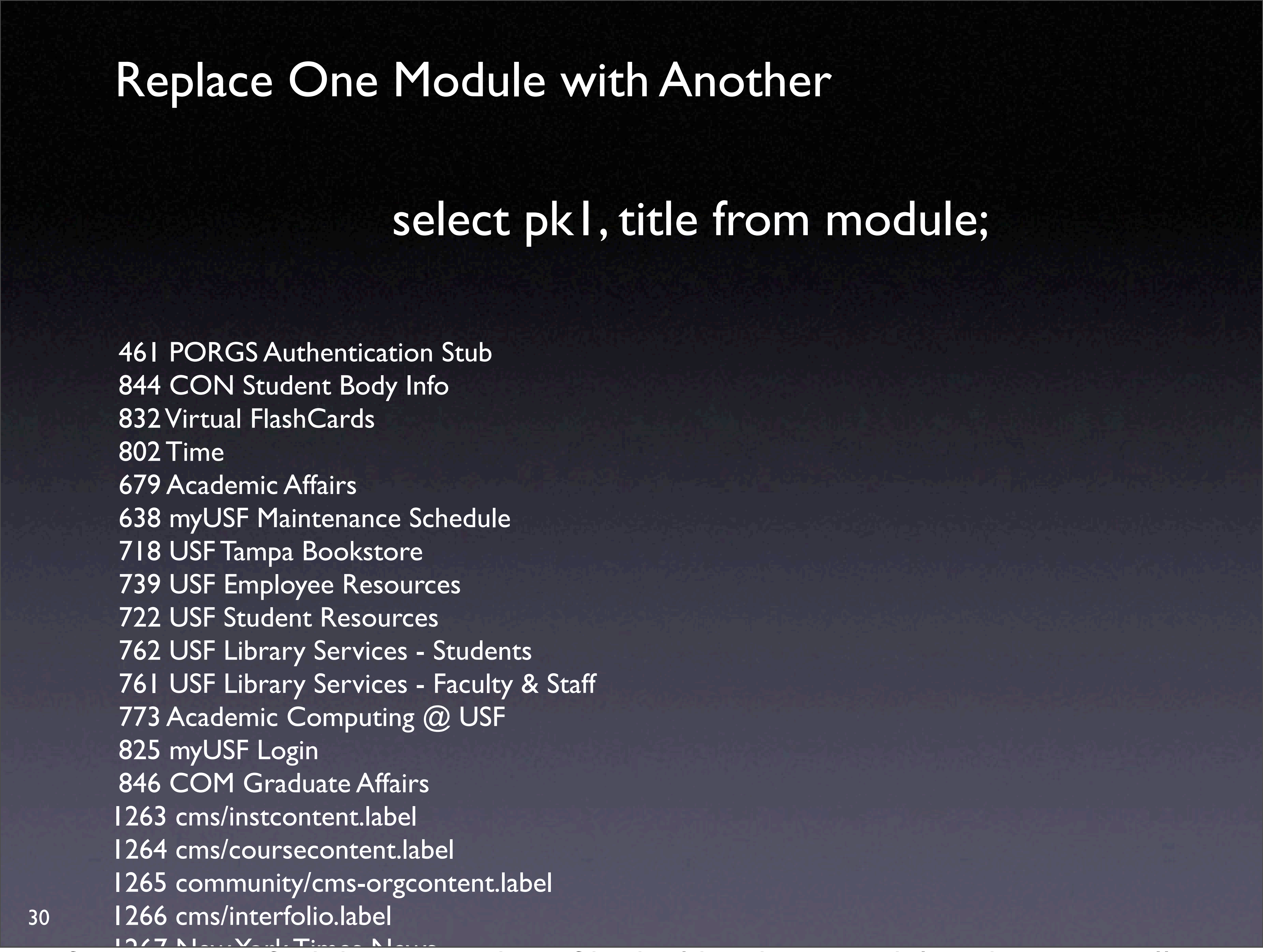
If you wanted to disable the tool for a single course, make this minor change to specify the pk1 of your course.

Replace One Module with Another

Here’s my situation. We wrote a great bookmarks building block portal module which lets you save your web bookmarks in a Blackboard module. But our module does much more than blackboards module. Our’s allows folders of bookmarks, Makes it possible to reorder bookmarks and move them around, and to import bookmarks from Internet Explorer or Mozilla. Very nice.

What we wanted to do was to identify people who were using the old bookmarks module, and replace that module with our module. For our project we went one step further and actually read the saved bookmarks from the old module, and inserted them into the new module for the user while we were replacing the module for them.

The steps needed to do this too specific to our bookmarks project, so I’m going to just cover the steps needed to replace the modules.



Replace One Module with Another

select pk1, title from module;

```
 461 PORGS Authentication Stub
 844 CON Student Body Info
 832 Virtual FlashCards
 802 Time
 679 Academic Affairs
 638 myUSF Maintenance Schedule
 718 USF Tampa Bookstore
 739 USF Employee Resources
 722 USF Student Resources
 762 USF Library Services - Students
 761 USF Library Services - Faculty & Staff
 773 Academic Computing @ USF
 825 myUSF Login
 846 COM Graduate Affairs
1263 cms/instcontent.label
1264 cms/coursecontent.label
1265 community/cms-orgcontent.label
1266 cms/interfolio.label
```

We first step is to find the primary key of both old and new module. This query will simply list all modules and their primary key. Run it and you'll get a very long list, similar to this. Scan the list and find the old module and the new, write down the primary keys.

Replace One Module with Another

```
update module_layout
set module_pk1=?
where module_pk1=?
```

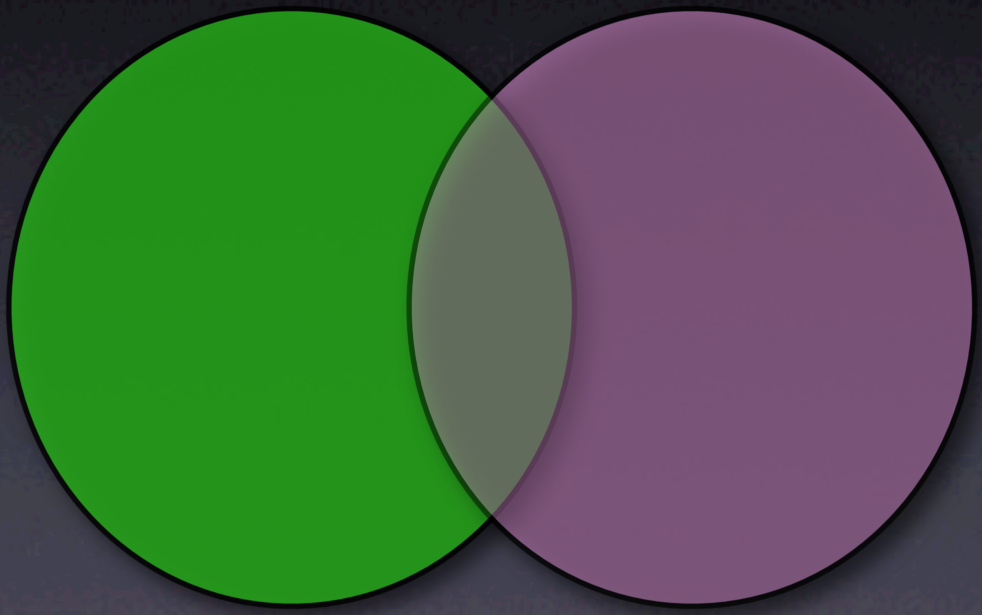
With old and new in hand, swapping them is straightforward. How it works is that each user on the system has one or more portal tabs which contain their modules. Each tab has a layout that is unique to that tab. The layout tells us which modules are on that portal page, and where on the page they are located. in the database, it is the module_layout table which relates individual modules to a particular tab layout.

For this project, we simply want to identify all layouts that contain the old module primary key, and replace it with the new module primary key. The end result of this query is that the next time the user accesses the tab, the old module will have been replaced with the new. The position and column will remain the same.

Identify Students Not Enrolled In Any Courses

This problem came about while reading the BBADMIN mailing lists. One member of the list asked how to generate a list of all users who were not enrolled in any courses. I don't remember exactly why she was asking for this information, but the lure of a new SQL challenge was too much to pass on.

Identify Students Not Enrolled In Any Courses



union
union all
minus
intersect

I chose this query because it lets me use the often underutilized SQL set operators. Set operators, as their name implies, work on sets of rows rather than on individual rows. They’re really useful to have in your arsenal of SQL tricks, as we’ll see with this problem.

Identify Students Not Enrolled In Any Courses

```
select ui.pk1 from users ui
```

If we’re going to use a set operator, then we’ll need a couple of sets.
First find the set of all users in the system, which we get by selecting the pk1 from the users table.

Identify Students Not Enrolled In Any Courses

```
select distinct(cu.users_pk1)
from course_users cu;
```

Next we get the set of users enrolled in one or more courses. Selecting all user pk1's from course users will accomplish this, but we'd ideally like to remove duplicates from this list, and for this we use distinct. Now we have a unique set of users involved in courses.

Identify Students Not Enrolled In Any Courses

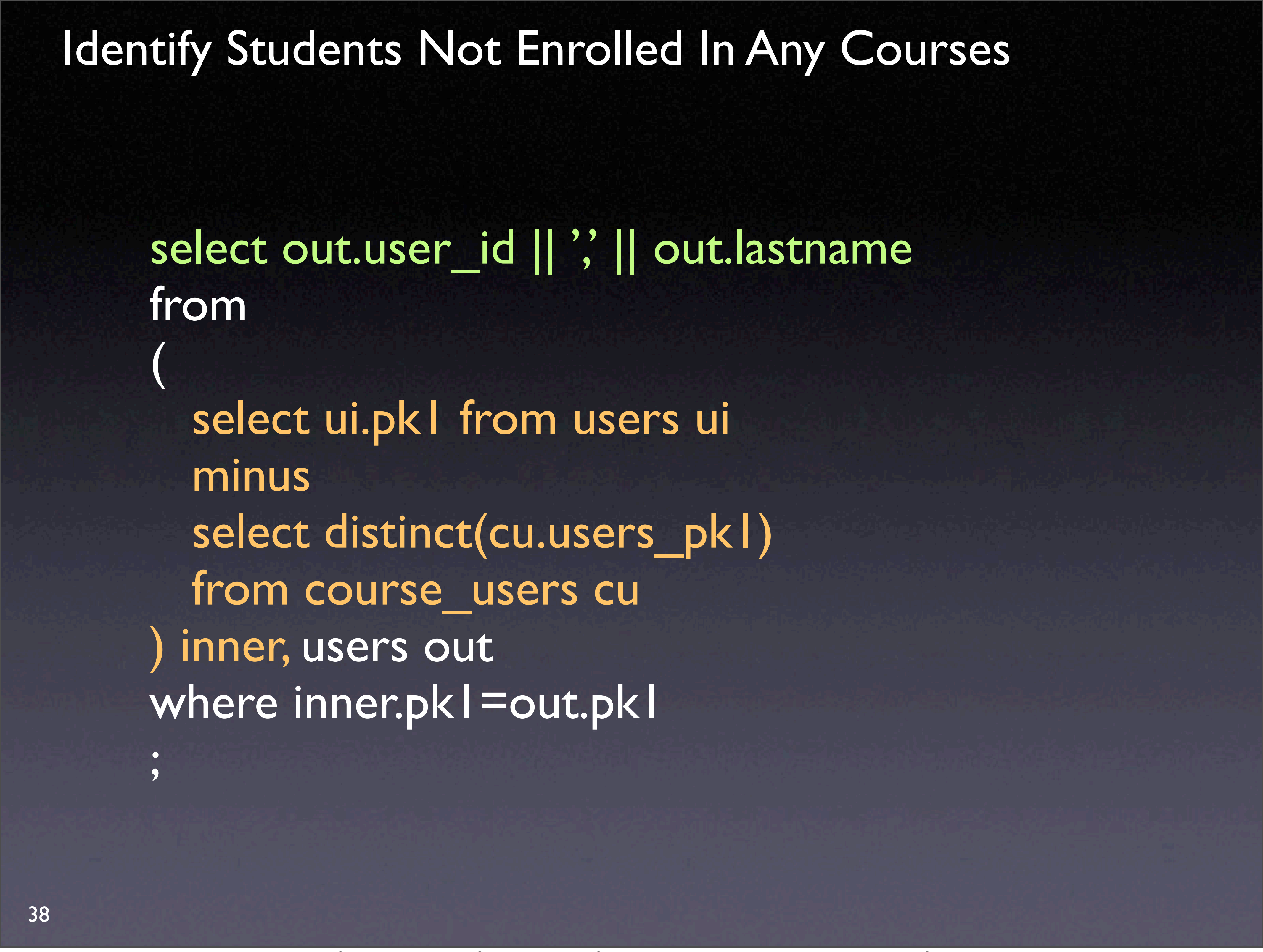
```
select ui.pk1 from users ui
minus
select distinct(cu.users_pk1)
from course_users cu;
```

Now we subtract the second set from the first set. What's left are all users not enrolled with any courses. Exactly what we are looking for.

Identify Students Not Enrolled In Any Courses

```
select out.email, out.firstname, out.lastname
from
(
   select ui.pk1 from users ui
   minus
   select distinct(cu.users_pk1)
   from course_users cu
) inner, users out
where inner.pk1=out.pk1
;
```

Now that we have the list of users, we can simply output the name and email address from the users table, and send them some marketing email. Let the students know that we have new courses they can register for, use the information for marketing purposes.



Identify Students Not Enrolled In Any Courses

```
select out.user_id || ’,’ || out.lastname
from
(
   select ui.pk1 from users ui
   minus
   select distinct(cu.users_pk1)
   from course_users cu
) inner, users out
where inner.pk1=out.pk1
;
```

Or you could write the file in the format of batch remove. In this format, it’s really easy to cut and paste into a text file and use Blackboard’s Batch Remove Users to remove these users from the system. Here we concatenate the user_id with a comma and the lastname, which is exactly the format that batch_remove users required

Change a Lot of Passwords all at Once

At South Florida, We actually run 2 instances of Blackboard, one of which is a Basic system. We use this to support groups that can't get into our main system as they don't have accounts in Banner.

A few months ago, I created a batch of about 40 new students, and emailed the instructor a file with the logins and passwords. Short story is that the file was lost, and so we had to either reset the passwords or recreate the accounts.

That got me looking into how blackboard stores passwords, and how to use SQL to change them myself.
Which led me to this solution.

Change a Lot of Passwords all at Once

ORACLE

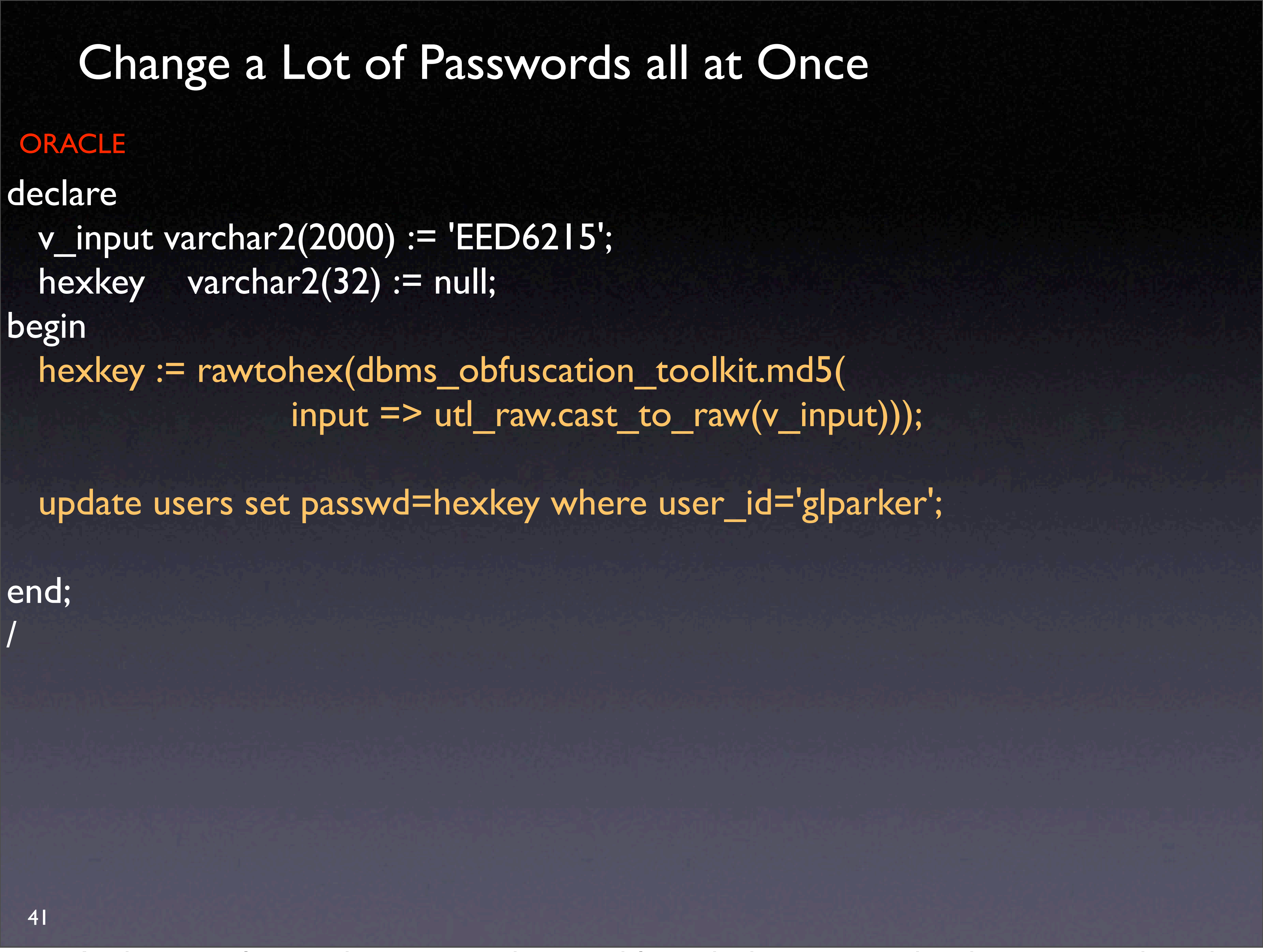
```
hexkey := rawtohex(dbms_obfuscation_toolkit.md5(
                        input => utl_raw.cast_to_raw(v_input)));

update users set passwd=hexkey where user_id='glparker';
```

Some caveats.
First, these next bits of SQL only work for Oracle. MSSQL I'm sure has equivalent functions but I don't know what they are.

Second is that this only works if you are storing passwords locally, and if you are not using SSL. Passwords stored on a regular system are stored in an MD5 encoded format, which passwords under SSL are stored in a Base64 encoding. We're interested for now in the MD5 format.

For this to work we use the Oracle function dbms_obfuscation_toolkit.md5, which takes as input a raw value. We can take a normal varchar and cast it to raw using the utl_raw function cast_to_raw. Once we have our MD5 string, we simply update the password field in the users table, and our password is reset.



Change a Lot of Passwords all at Once

ORACLE

```
declare
   v_input varchar2(2000) := 'EED6215';
   hexkey     varchar2(32) := null;
begin
   hexkey := rawtohex(dbms_obfuscation_toolkit.md5(
                         input => utl_raw.cast_to_raw(v_input)));

   update users set passwd=hexkey where user_id='glparker';

end;
/
```

Here’s the complete code.   We need a variable with the password to be encrypted, and variable to hold the resulting MD5 hash. It’s a really simply script for such a powerful tool.

Notice here how I’m updating a single user with my new password, in this case the user glparker. This update is the real flexible part of this method.

Change a Lot of Passwords all at Once

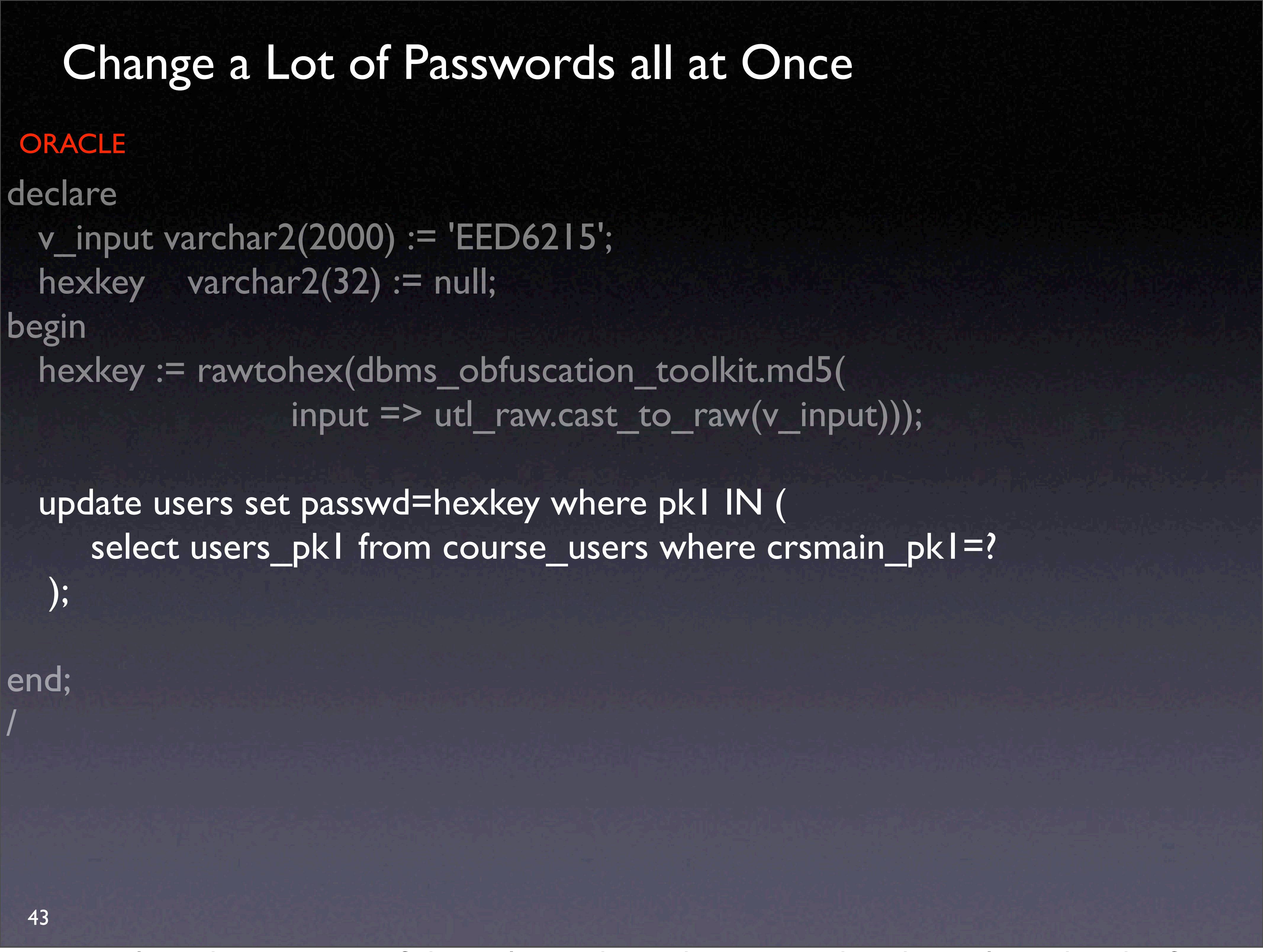
ORACLE

```
declare
   v_input varchar2(2000) := 'EED6215';
   hexkey     varchar2(32) := null;
begin
   hexkey := rawtohex(dbms_obfuscation_toolkit.md5(
                           input => utl_raw.cast_to_raw(v_input)));

   update users set passwd=hexkey where user_id IN ('glparker','santo','edgaray');

end;
/
```

I can replace the single user_id with an IN list and update a bunch of single users in one execution.
-- or --



Change a Lot of Passwords all at Once

ORACLE

```
declare
   v_input varchar2(2000) := 'EED6215';
   hexkey     varchar2(32) := null;
begin
   hexkey := rawtohex(dbms_obfuscation_toolkit.md5(
                          input => utl_raw.cast_to_raw(v_input)));

   update users set passwd=hexkey where pk1 IN (
        select users_pk1 from course_users where crsmain_pk1=?
    );

end;
/
```

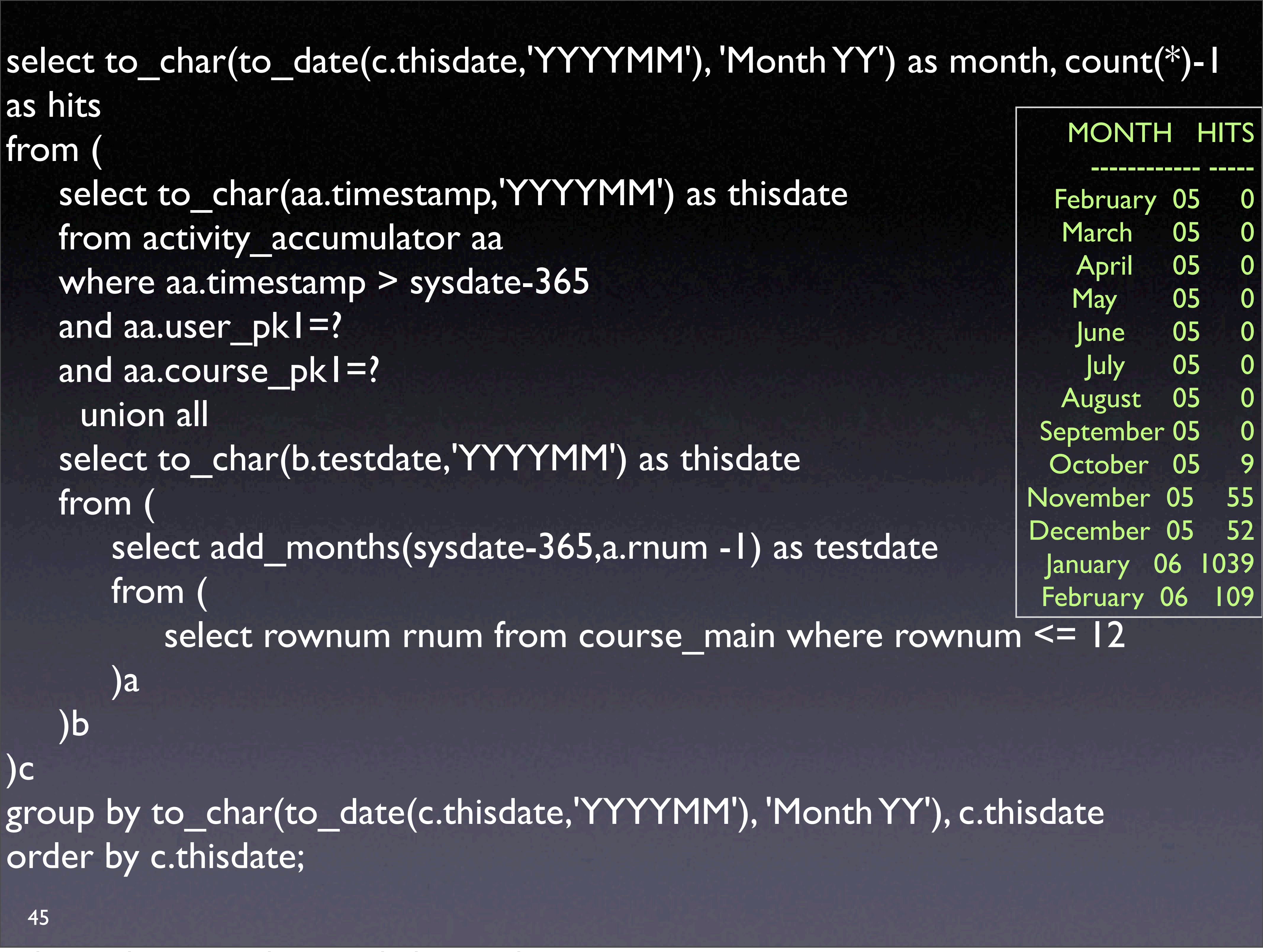
I can replace the contents of the IN list with a subquery, and update a large batch of users, such as all users in a particular course; or all users on the entire system, all users whose last name starts with ‘G’, all users who haven’t logged in in two weeks.

The real take-away from this example that that your database provides a huge toolchest of functions, and your shouldn’t be afraid to open the SQL manual for your database and see what else they provide. You are not necessarily limited to Basic SQL.

Student Course Access by Month

Last problem is my piez-de-resistance. This is a great query. I originally wrote this about a year ago as part of a building block we were working on called Student Performance Assistant.

One section of this building block was a report card for students which showed their grades, discussion board activity, and course activity. In particular, it showed a count of how times they clicked inside the course, grouped by month. So for the previous twelve months, the instructor would know how many times a student clicked inside their course.



```
select to_char(to_date(c.thisdate,'YYYYMM'), 'Month YY') as month, count(*)-1
as hits
from (
    select to_char(aa.timestamp,'YYYYMM') as thisdate
    from activity_accumulator aa
    where aa.timestamp > sysdate-365
    and aa.user_pk1=?
    and aa.course_pk1=?
      union all
    select to_char(b.testdate,'YYYYMM') as thisdate
    from (
        select add_months(sysdate-365,a.rnum -1) as testdate
        from (
            select rownum rnum from course_main where rownum <= 12
        )a
    )b
)c
group by to_char(to_date(c.thisdate,'YYYYMM'), 'Month YY'), c.thisdate
order by c.thisdate;
```

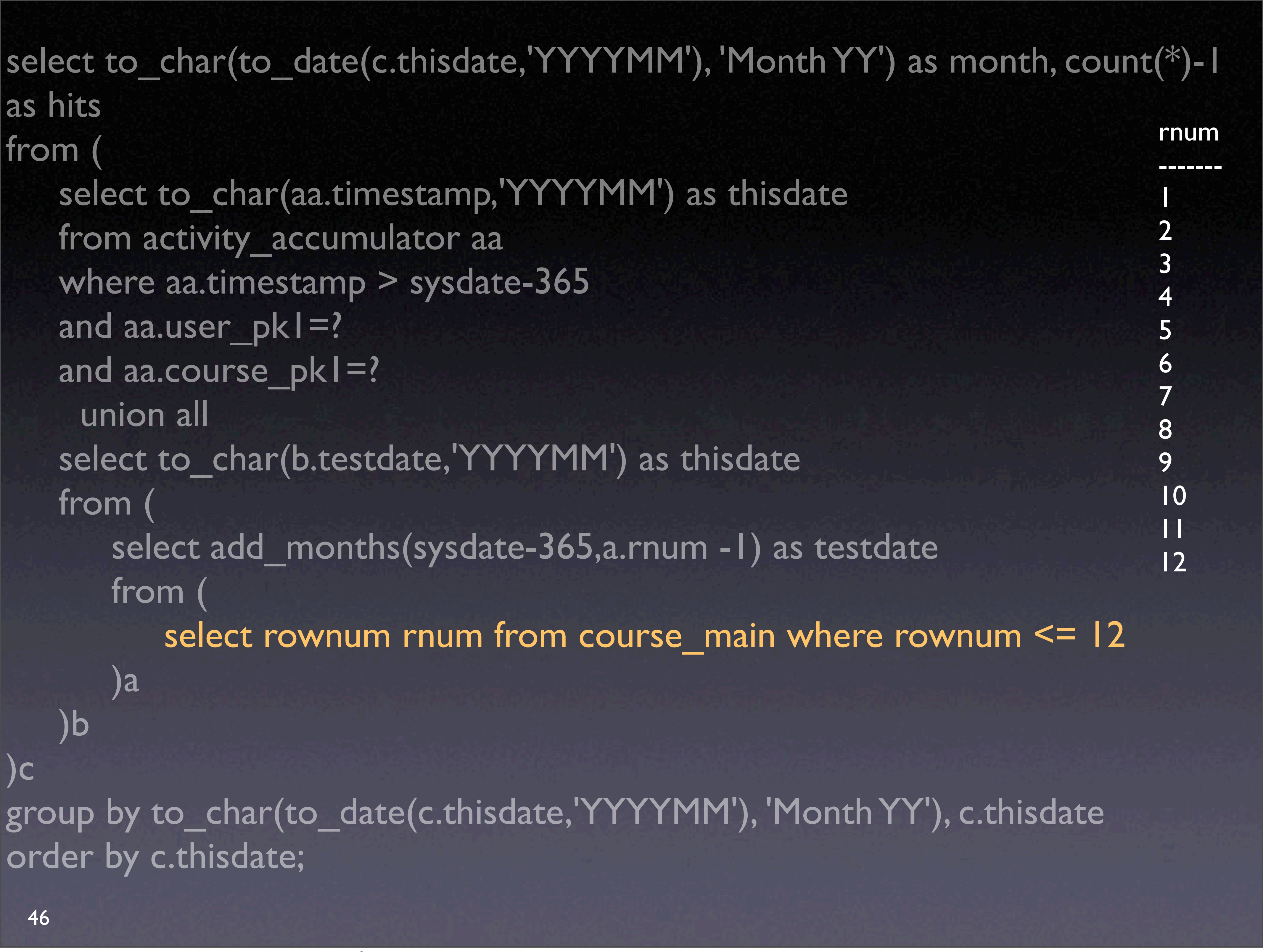
```
      MONTH    HITS
 ------------ -----
  February 05     0
  March    05     0
  April    05     0
  May      05     0
  June     05     0
  July     05     0
  August   05     0
  September 05    0
  October  05     9
  November 05    55
  December 05    52
  January  06  1039
  February 06   109
```

This is the query that I ended up with.

Don't get scared, we're going to break this down over the next 5 slides. I'm showing you the entire query upfront, so you know where we are headed. But don't try to hard to make sense of what's on screen just yet.

Also, pay particular attention to the green text on the right. I'll keep a box on the right side of the screen showing what the output from the query looks like at that stage of the query. That is, if we were to run that portion of the query right now, the box on the right will show what would be returned.

There's a lot of neat SQL tricks here to see... Let's get started.



```
select to_char(to_date(c.thisdate,'YYYYMM'), 'Month YY') as month, count(*)-1
as hits
from (
    select to_char(aa.timestamp,'YYYYMM') as thisdate
    from activity_accumulator aa
    where aa.timestamp > sysdate-365
    and aa.user_pk1=?
    and aa.course_pk1=?
      union all
    select to_char(b.testdate,'YYYYMM') as thisdate
    from (
        select add_months(sysdate-365,a.rnum -1) as testdate
        from (
            select rownum rnum from course_main where rownum <= 12
        )a
    )b
)c
group by to_char(to_date(c.thisdate,'YYYYMM'), 'Month YY'), c.thisdate
order by c.thisdate;
```

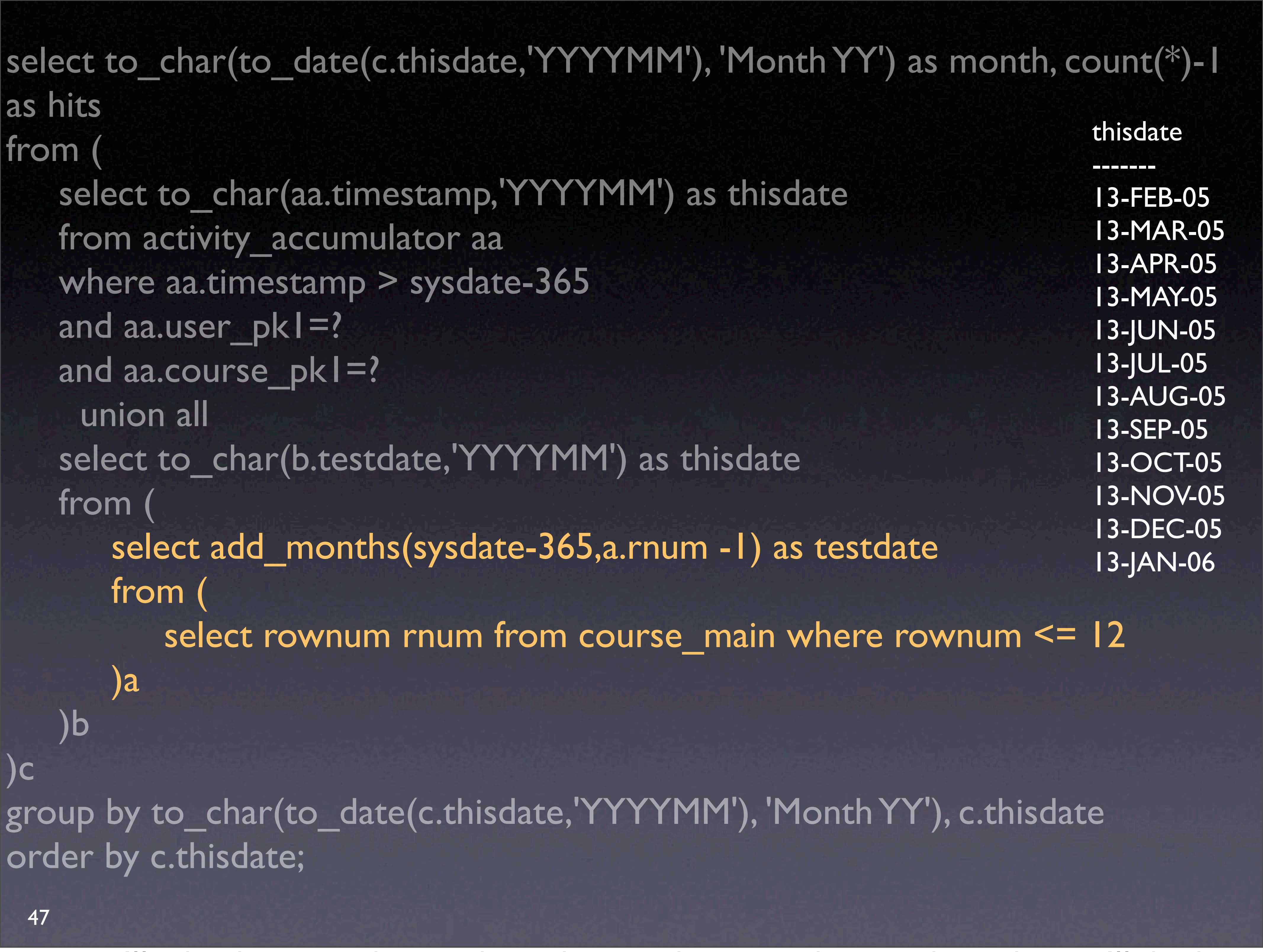
```
rnum
-------
1
2
3
4
5
6
7
8
9
10
11
12
```

We'll build the query up from the inside out. The lines in yellow will show what we are currently spoghtlighting.

To start, we'll select 12 rownum's from a table with at least twelve rows. rownum is a virtual column used by Oracle, and I believe it's also available for MSSQL. Rownum is a number showing which row number you are looking at among a list of rows in a result set.

Also notice that I select the rownum from the course main table. It doesn't matter where you get your rownums from so long as the table you select it has at least 12 rows.

Why do this? These 12 numbers will form the basis of the 12 months used in the final report.



```
select to_char(to_date(c.thisdate,'YYYYMM'), 'Month YY') as month, count(*)-1
as hits
from (
    select to_char(aa.timestamp,'YYYYMM') as thisdate
    from activity_accumulator aa
    where aa.timestamp > sysdate-365
    and aa.user_pk1=?
    and aa.course_pk1=?
      union all
    select to_char(b.testdate,'YYYYMM') as thisdate
    from (
        select add_months(sysdate-365,a.rnum -1) as testdate
        from (
            select rownum rnum from course_main where rownum <= 12
        )a
    )b
)c
group by to_char(to_date(c.thisdate,'YYYYMM'), 'Month YY'), c.thisdate
order by c.thisdate;
```

```
thisdate
-------
13-FEB-05
13-MAR-05
13-APR-05
13-MAY-05
13-JUN-05
13-JUL-05
13-AUG-05
13-SEP-05
13-OCT-05
13-NOV-05
13-DEC-05
13-JAN-06
```

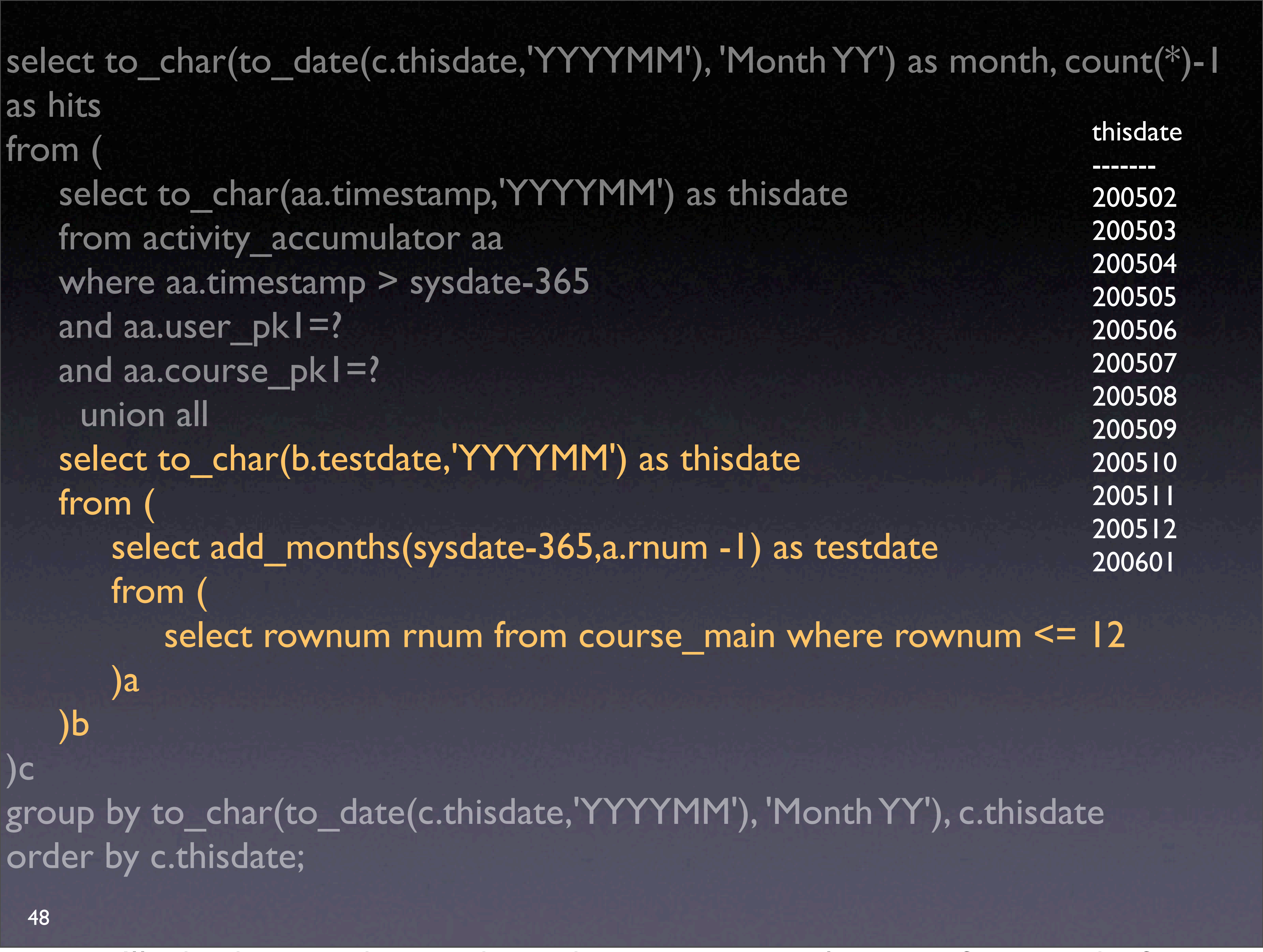
Now we'll take those numbers 1 through 12 and convert them to dates, but we'll do this conversion in a very specific way. In particular I want to convert these numbers to todays day of the month, for each of the last twelve months. That is, I want each number to represent each month for the last 12 months.

We're going to do this with the add_months function. This function takes two arguments, first a date, and second the number of months you want to add to that date. I'm going to go back one year, which is sysdate minus 365 days, and then add months one at a time until I've added 12 months.

So we look at the first row returned, which was the number 1. We subtract one from that number, which is zero, and then add that many months to sysdate-365. The result is sysdate-365 or a year ago today. We see that in the first row in the result set on the right. February of 2005.

We continue with the second row, which was 2. 2 minus 1 is 1, so we add 1 months to sysdate-365, which was 11 months ago or March 2005.

And so on. 3 becomes April 2005, 4 becomes May 2005, right on up to the current month. Everybody cool with this so far. It only gets better from here.

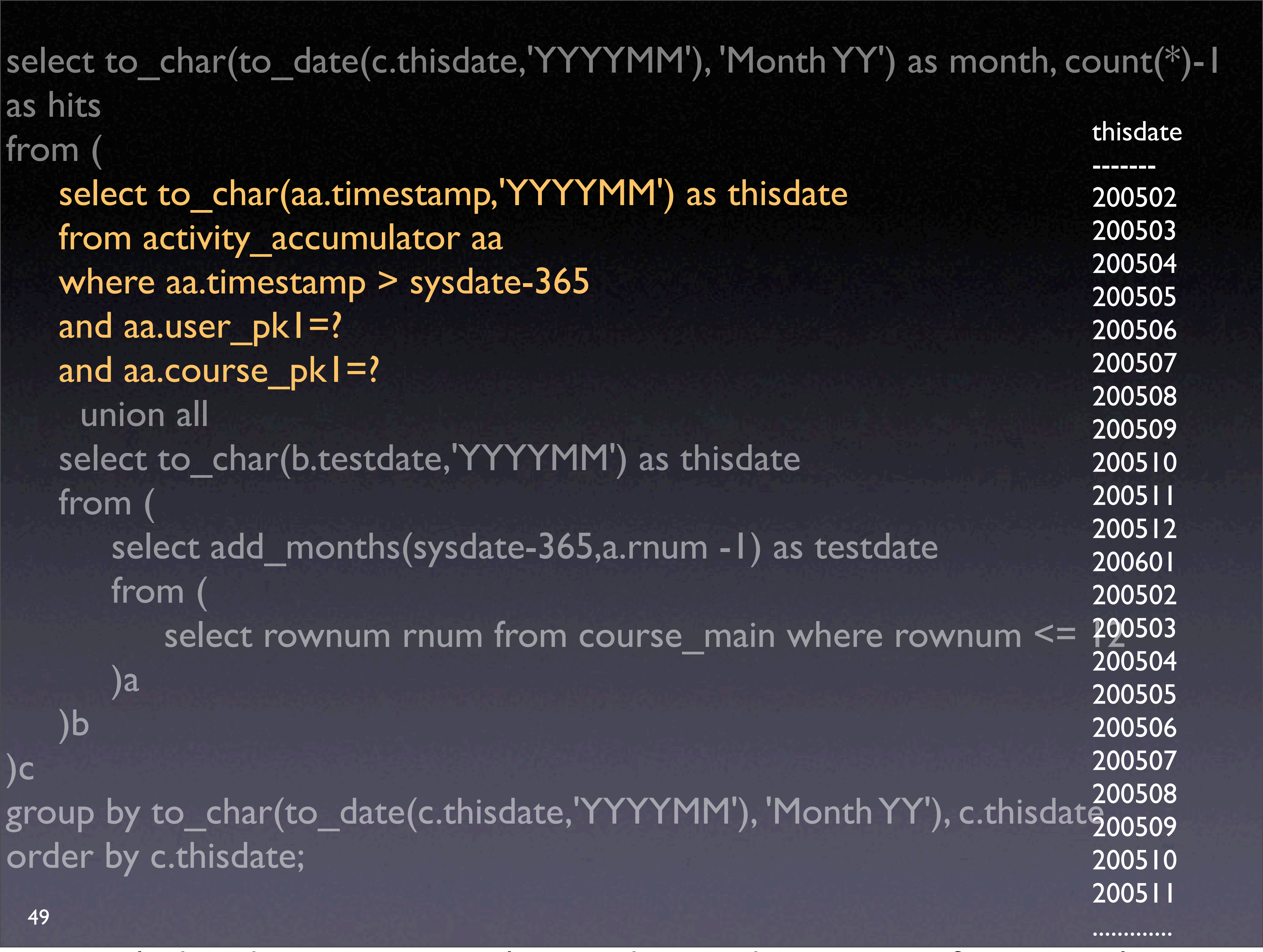


```
select to_char(to_date(c.thisdate,'YYYYMM'), 'Month YY') as month, count(*)-1
as hits
from (
    select to_char(aa.timestamp,'YYYYMM') as thisdate
    from activity_accumulator aa
    where aa.timestamp > sysdate-365
    and aa.user_pk1=?
    and aa.course_pk1=?
      union all
    select to_char(b.testdate,'YYYYMM') as thisdate
    from (
        select add_months(sysdate-365,a.rnum -1) as testdate
        from (
            select rownum rnum from course_main where rownum <= 12
        )a
    )b
)c
group by to_char(to_date(c.thisdate,'YYYYMM'), 'Month YY'), c.thisdate
order by c.thisdate;
```

```
thisdate
-------
200502
200503
200504
200505
200506
200507
200508
200509
200510
200511
200512
200601
```

New we’ll take those 12 dates and cast them into a particular string format. The format here is the 4 digit year followed by the 2 digit month.

Casting them to a specific format now makes it easier to group and count them later on. Lets set these aside and look to the next section.

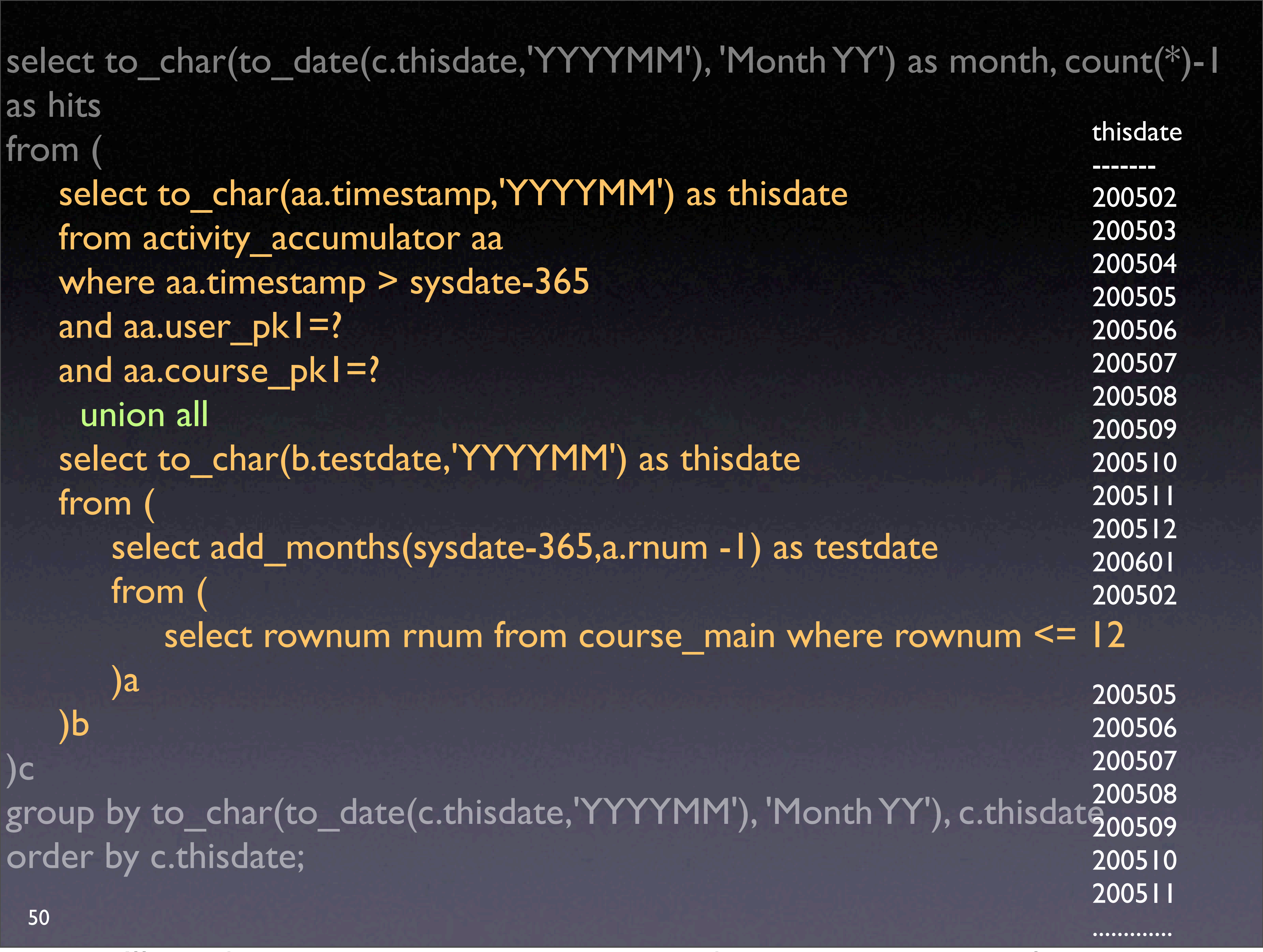


```
select to_char(to_date(c.thisdate,'YYYYMM'), 'Month YY') as month, count(*)-1
as hits
from (
    select to_char(aa.timestamp,'YYYYMM') as thisdate
    from activity_accumulator aa
    where aa.timestamp > sysdate-365
    and aa.user_pk1=?
    and aa.course_pk1=?
      union all
    select to_char(b.testdate,'YYYYMM') as thisdate
    from (
        select add_months(sysdate-365,a.rnum -1) as testdate
        from (
            select rownum rnum from course_main where rownum <= 12
        )a
    )b
)c
group by to_char(to_date(c.thisdate,'YYYYMM'), 'Month YY'), c.thisdate
order by c.thisdate;
```

```
thisdate
-------
200502
200503
200504
200505
200506
200507
200508
200509
200510
200511
200512
200601
200502
200503
200504
200505
200506
200507
200508
200509
200510
200511
............
```

Here we look in the activity accumulator, and return the timestamp for a particular user in a particular course.   This will hopefully return a lot of rows, indicating a lot of activity in the course.

We convert the timestamp into the same string format we used on the previous screen. What we end up with is a long list of date strings, one for each page clicked by a student in a course. This is our users activity, what we are ultimately interested in counting.

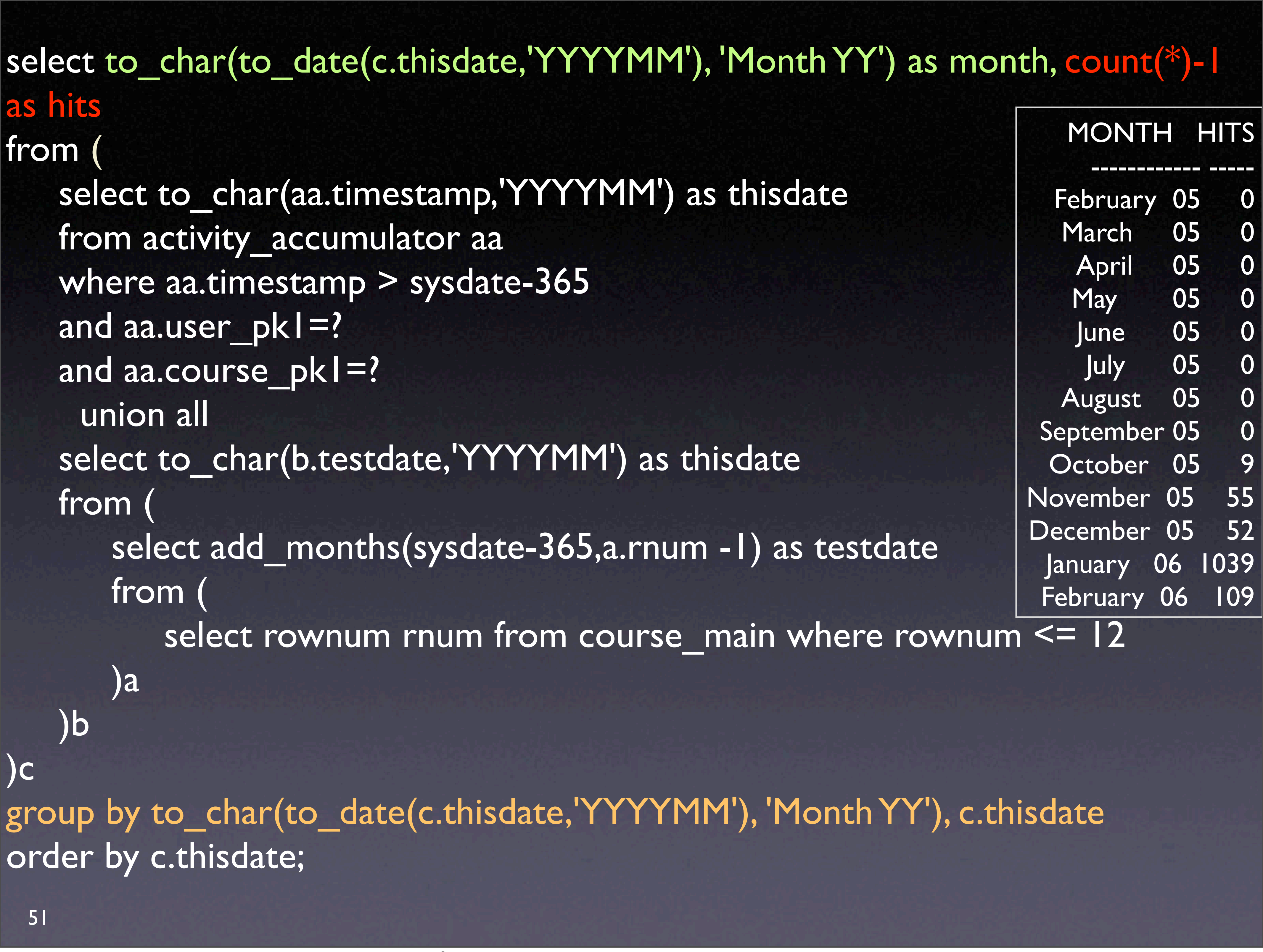


```
select to_char(to_date(c.thisdate,'YYYYMM'), 'Month YY') as month, count(*)-1
as hits
from (
    select to_char(aa.timestamp,'YYYYMM') as thisdate
    from activity_accumulator aa
    where aa.timestamp > sysdate-365
    and aa.user_pk1=?
    and aa.course_pk1=?
      union all
    select to_char(b.testdate,'YYYYMM') as thisdate
    from (
        select add_months(sysdate-365,a.rnum -1) as testdate
        from (
            select rownum rnum from course_main where rownum <= 12
        )a
    )b
)c
group by to_char(to_date(c.thisdate,'YYYYMM'), 'Month YY'), c.thisdate
order by c.thisdate;
```

```
thisdate
-------
200502
200503
200504
200505
200506
200507
200508
200509
200510
200511
200512
200601
200502

200505
200506
200507
200508
200509
200510
200511
.............
```

Now we'll use the UNION ALL set operator to join these two sets into one larger one. We now have all our students activity along with the 12 months we originally crafted.
If this seems like a lot of extra effort to get the student activity, well it is, but it's necessary to achieve the desired result, which we get on the final slide



```
select to_char(to_date(c.thisdate,'YYYYMM'), 'Month YY') as month, count(*)-1
as hits
from (
    select to_char(aa.timestamp,'YYYYMM') as thisdate
    from activity_accumulator aa
    where aa.timestamp > sysdate-365
    and aa.user_pk1=?
    and aa.course_pk1=?
      union all
    select to_char(b.testdate,'YYYYMM') as thisdate
    from (
        select add_months(sysdate-365,a.rnum -1) as testdate
        from (
            select rownum rnum from course_main where rownum <= 12
        )a
    )b
)c
group by to_char(to_date(c.thisdate,'YYYYMM'), 'Month YY'), c.thisdate
order by c.thisdate;
```

```
   MONTH      HITS
 ------------ -----
 February  05     0
 March     05     0
 April     05     0
 May       05     0
 June      05     0
 July      05     0
 August    05     0
 September 05     0
 October   05     9
 November  05    55
 December  05    52
 January   06  1039
 February  06   109
```

Finally, we take the large set of date strings, group them, and count the groups, then adjust the counts to make them correct.

First the grouping. This is the yellow clause on the bottom of the screen. Here we take the large set and group them first by the date format we will eventually output, but also by the raw date string returned by the inner query.

Next we format the month and year so that it looks like a month and year, and not like some Banner code. This is shown in the green section at the top. What I'm doing here is taking the date string I received from the inner query, converting it back into a date, then reconverting the date back to a string in a more human readable format.

Finally we subtract one from each months count to remove the effect of the original twelve months. This is shown in the red clause on the top of the screen. So what we did was add one to each months count as part of the grouping, then remove that one for the report.

Why add then subtract? If we did not do this, then the months like April and June would have 0 records in the activity_accumulator to read and count. But worse than 0 records, this would actually be NULL number of records, and the NULL records would never be printed. There's no way to get a row for June 05 if there are no records in June to read and count. By forcing there to be at least one record for June, we guarantee that we include June in the final report.

So... What do you think? Is that an impressive query or what? Any questions on this query? . . . .

Beyond SQL

OK, Let’s look beyond the sql interface, and see what we can do with our new found knowledge. Here’s where we can apply our knowledge of Blackboard and develop tools to make our jobs as admins easier and more productive. Plus we end up looking really cool when we turn around a problem in minutes instead of hours or days.

Data Mining

Data mining. This is a big topic. Take the section we just looked at on blackboard course usage and extend it to look at every possible nuance of your system. All sorts of usage information can be gleaned from the database, along with demographics of the people using certain pieces. Some questions you might be able to answer might include :
  Do instructors over age 40 tend to use the dropbox or the assignmetns tool?
  Can we correlate training classes we offer with tool usage in Blackboard.
  Do students do better on tests in courses that use more than 5 tools?
If you can ask the question, then the database likely has the answers.

I gave a talk on data mining Blackboard at the developers conference in 2005. You can get the slides from my web site.
Also on the site is some perl code you can run to generate a very nice report of Blackboard usage on your own system. It's Very cool. Unfortunately, it's Unix only, but it's free so if someone wants to port it to Windows, feel free.

Modifying Your Local Schema

Once you've studied your particular system for a while, you might start to notice some of the design decisions that were made by Blackboard. Blackboard built and tuned the database to work with their queries, and built it pretty well for the most part.

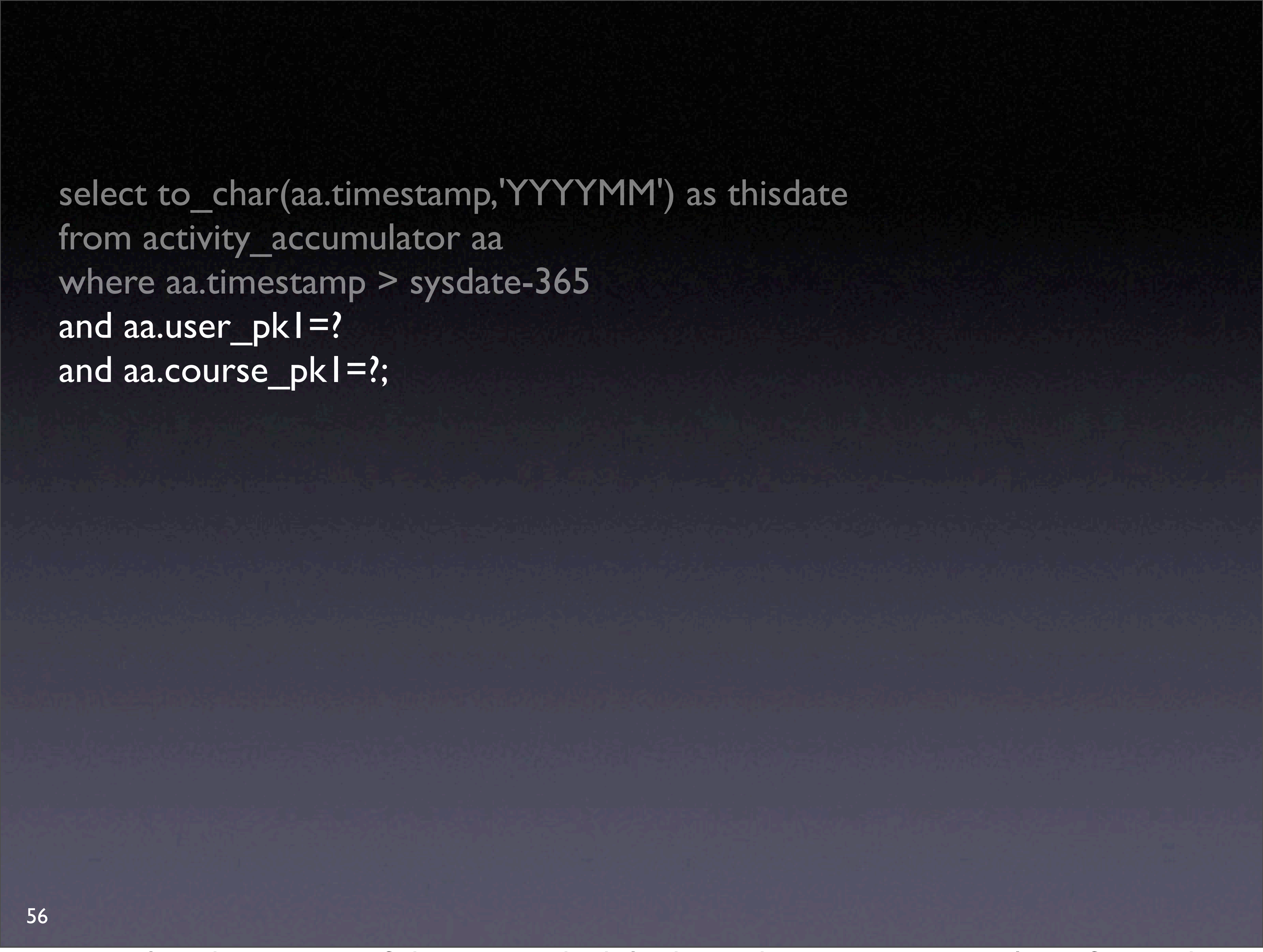
However, you are doing things in SQL that Blackboard never envisioned, so your queries might not be running as well as they possibly could. It's certainly possible to make changes to improve your queries, or to improve Blackboard queries if you happen to find a deficiency.

About a year ago, I found a database problem. The symptoms were that Course Usage Statistics report that instructors can run from the control panel of there course would not finish. The browser would timeout. In particular reports that made use of item tracking would not complete. I eventually found that queries were being run against the activity_accumulator and were specifying the content_pk1 column as the only criteria. It turns out that there was no index on the content_pk1 column, so Oracle's only other option was a full table scan of the activity_accumulator. Adding an index to the content_pk1 column reduced the time needed to run reports from hours down to seconds.

Blackboard added the index to the column in Blackboard 7, but if you think you are missing this index, you can contact Blackboard support for instructions adding it to older versions.

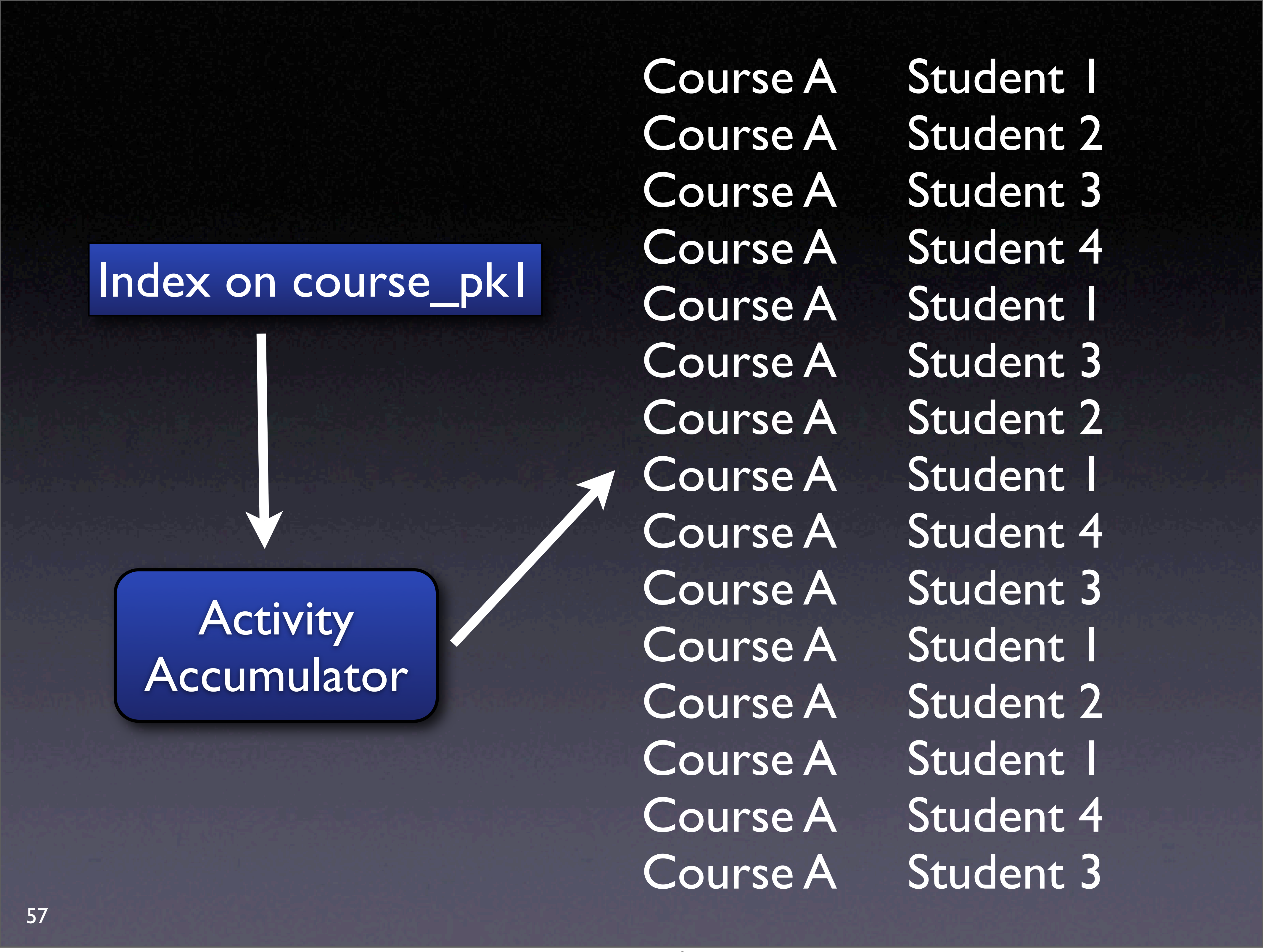
```
select to_char(to_date(c.thisdate,'YYYYMM'), 'Month YY') as month, count(*)-1
as hits
from (
     select to_char(aa.timestamp,'YYYYMM') as thisdate
     from activity_accumulator aa
     where aa.timestamp > sysdate-365
     and aa.user_pk1=?
     and aa.course_pk1=?
       union all
     select to_char(b.testdate,'YYYYMM') as thisdate
     from (
          select add_months(sysdate-365,a.rnum -1) as testdate
          from (
               select rownum rnum from course_main where rownum <= 12
          )a
     )b
)c
group by to_char(to_date(c.thisdate,'YYYYMM'), 'Month YY'), c.thisdate
order by c.thisdate;
```

But, let’s bring back the uber query from before. On a stock blackboard system, This query runs pretty well in 10-15 seconds, but it could be made to run better.



```
select to_char(aa.timestamp,'YYYYMM') as thisdate
from activity_accumulator aa
where aa.timestamp > sysdate-365
and aa.user_pk1=?
and aa.course_pk1=?;
```

In particular, the portion of the query which looks at the activity accumulator for student course activity. Notice how we are restricting the query to look only for a single user and a single course. This is a good and valid restriction, but there are some problems with this approach. Blackboard has an index on the users column and another index on the courses column, but because of the volume of data we are processing, the database is not able to optimize and use both of these indexes. Let's look at a picture



Course A	Student 1
Course A	Student 2
Course A	Student 3
Course A	Student 4
Course A	Student 1
Course A	Student 3
Course A	Student 2
Course A	Student 1
Course A	Student 4
Course A	Student 3
Course A	Student 1
Course A	Student 2
Course A	Student 1
Course A	Student 4
Course A	Student 3

Oracle will process the query and decide that it first needs to look at the index on course_pk1 for all rows in the activity accumulator with our course_pk1.
This returns all rows for our course, regardless of which student that row belongs to.

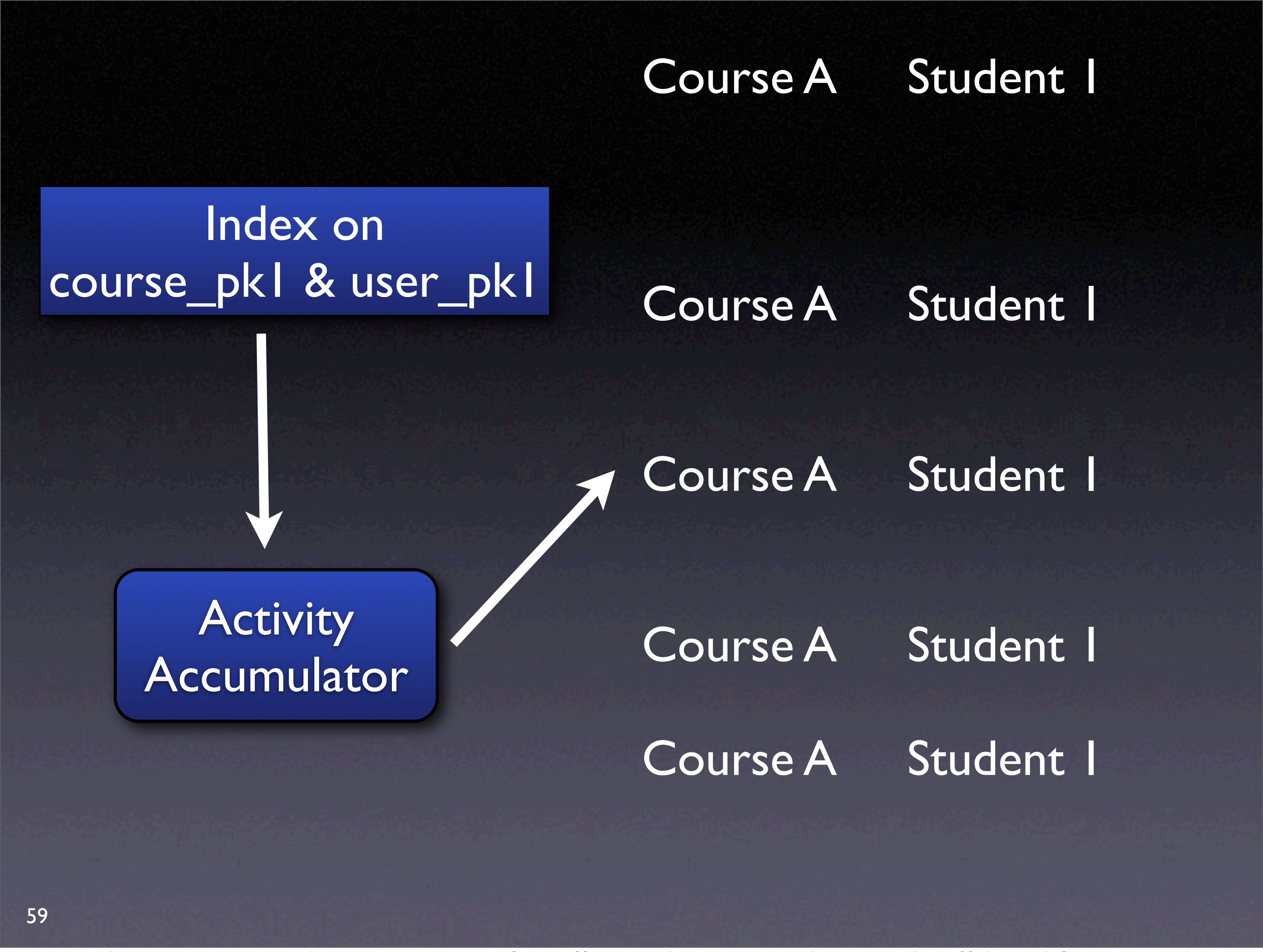
Oracles then has to iterate over all these rows looking for rows that belong to our particular student. This expensive and slow.

```
CREATE INDEX activity_accumulator_usf_ie1
ON activity_accumulator (course_pk1, user_pk1)
TABLESPACE bb_bb60_indx;
```

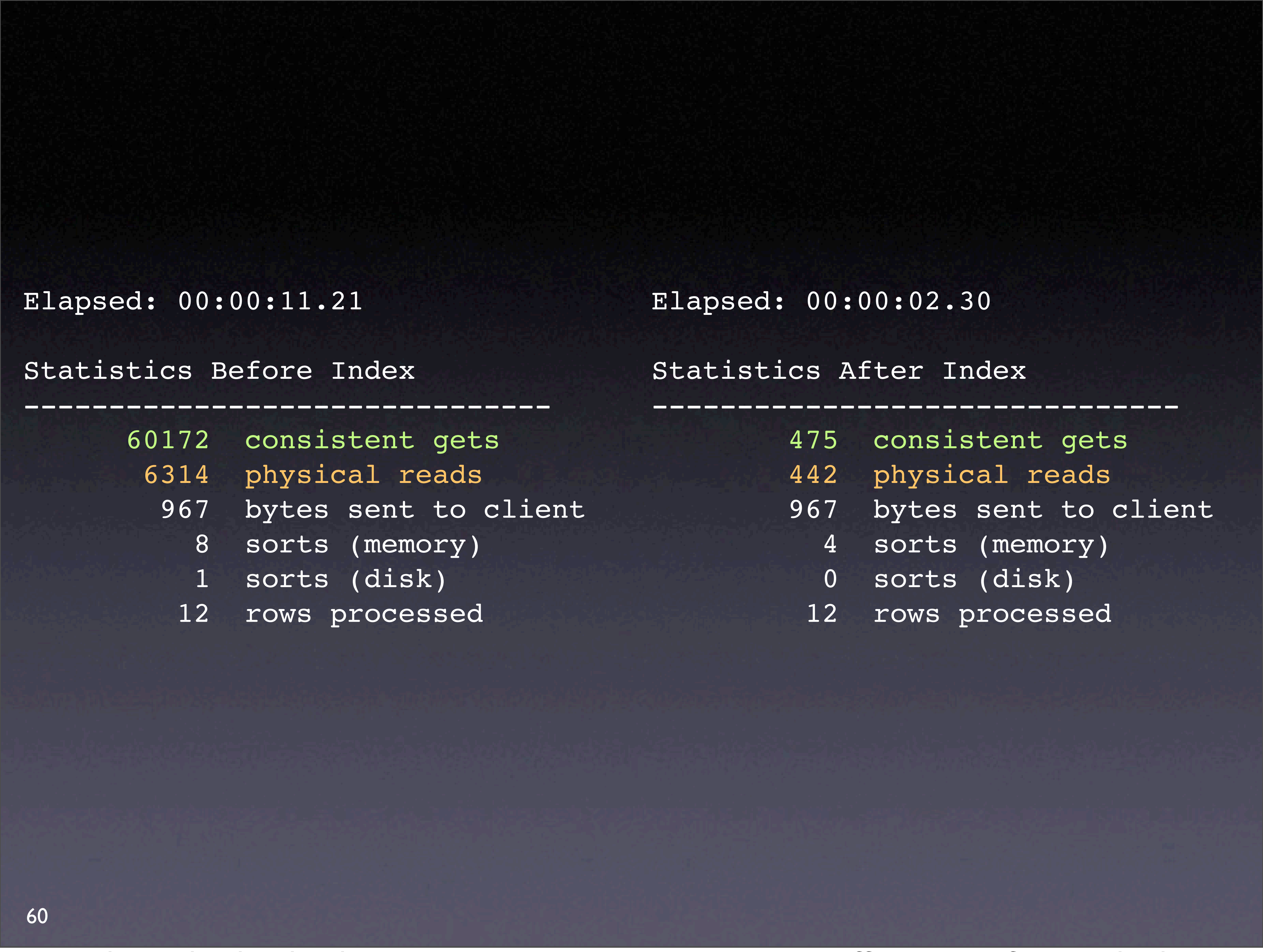
What we really need is a better index, which would allow Oracle to go directly to the rows that correspond to both our course AND our user.

The database could avoid that iteration if we had a different index that combined both the course and user.

We’ll create a third index on the combination of course and user, which will allow us to home in on exactly the course user pair we are interested in.



Now when we rerun the query, Oracle will use the new index, and will be able to read exactly the rows for our course and user. Oracle won't need to iterate over the rows, saving a lot of time.



```
Elapsed: 00:00:11.21                Elapsed: 00:00:02.30

Statistics Before Index             Statistics After Index
-------------------------------     -------------------------------
      60172  consistent gets                    475  consistent gets
       6314  physical reads                     442  physical reads
        967  bytes sent to client               967  bytes sent to client
          8  sorts (memory)                       4  sorts (memory)
          1  sorts (disk)                         0  sorts (disk)
         12  rows processed                      12  rows processed
```

I even brought the database statistics to prove it was more efficient. Before we had to read more than 64000 blocks, 6000 from disk, the rest from memory. After the index we had to read a meager 800 blocks. We also shaved 8 seconds off the total execution time 11 seconds down to 2. Not bad for a little index.

The best part is that other queries can take advantage of this index. Any query that looks at both a course and a user can take advantage of the new index.

However, nothing is ever free. We do pay a penalty for this index. Every insert, update, or deletion from the activity_accumulator incurs an additional operation as we have to maintain the new index. We have to use disk space to store the new index, and CPU time to keep the index up to date. This increases the time it takes to modify data in the activity_accumulator. If your database is already struggling with either CPU or Disk space, this could seriously impact the response time for your Blackboard user. Try it on a test system and see what happens.

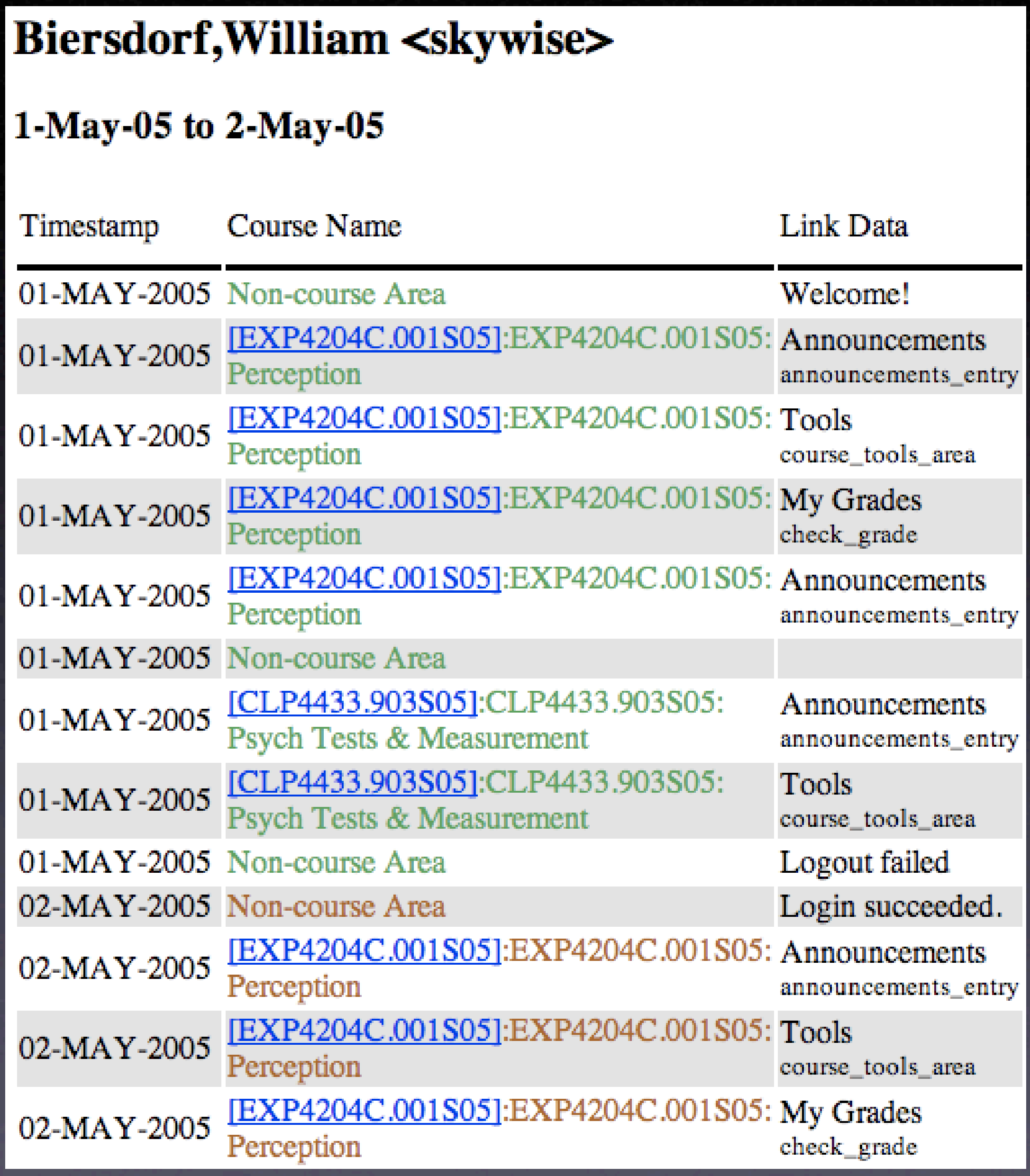
Write An External Application Against Your Blackboard Data

Another useful way to extend the data in your database is to write external application. Building block are a good example of such external applications, but you don't need to limit yourself to buildinglbocks. A web server with php also makes a great interface to your database.

User Activity

Big Brother is watching !

- Count unique institution roles and course roles
- Track session activity and duration
- Identify active courses
- activity_accumulator and sessions tables



Biersdorf,William <skywise>

1-May-05 to 2-May-05

Timestamp	Course Name	Link Data
01-MAY-2005	Non-course Area	Welcome!
01-MAY-2005	[EXP4204C.001S05]:EXP4204C.001S05: Perception	Announcements announcements_entry
01-MAY-2005	[EXP4204C.001S05]:EXP4204C.001S05: Perception	Tools course_tools_area
01-MAY-2005	[EXP4204C.001S05]:EXP4204C.001S05: Perception	My Grades check_grade
01-MAY-2005	[EXP4204C.001S05]:EXP4204C.001S05: Perception	Announcements announcements_entry
01-MAY-2005	Non-course Area	
01-MAY-2005	[CLP4433.903S05]:CLP4433.903S05: Psych Tests & Measurement	Announcements announcements_entry
01-MAY-2005	[CLP4433.903S05]:CLP4433.903S05: Psych Tests & Measurement	Tools course_tools_area
01-MAY-2005	Non-course Area	Logout failed
02-MAY-2005	Non-course Area	Login succeeded.
02-MAY-2005	[EXP4204C.001S05]:EXP4204C.001S05: Perception	Announcements announcements_entry
02-MAY-2005	[EXP4204C.001S05]:EXP4204C.001S05: Perception	Tools course_tools_area
02-MAY-2005	[EXP4204C.001S05]:EXP4204C.001S05: Perception	My Grades check_grade

Remember, with a simple query to the activity_accumulator table, we can see most every click a student makes in a course. This screen shows a sample output from a web based tool we developed to perform just such queries. We enter the course ID and user ID and a range of dates we are interested in, and a report comes back in very short order. We've even gone so far as to color code group the activity by individual sessions.

In this sample slide, we see that our student has entered the system on May 1, entered the course and saw the Announcement page, then clicked on Course Tools to check their grade, Clicked another page which took them back to the announcements. From here they clicked the courses tab to bring up their course list, and which they used to visit their other course. They looked at their announcements and clicked the tools area, but left before they could do anything else.
Finally, they came back the next day to check grades one more time.

What to do with this information? Besides spying on your students, you could do as we recently did, and used this tool to identify a student who was impersonating the instructor by sending emails with a forged header. We identified them by noticing a person making repeated roster lookups, and also noticing that the emails were sent within minutes of the roster lookups, and that the email were delivered in the same order that the roster returned the addresses, alphabetically by last name.

Conclusion

- Questions/Answers?
- Sample Code? Freebies?

Glen Parker
glparker@usf.edu

http://presentation.glenparker.net