



## Chapter 4 : Intermediate SQL

**Database System Concepts, 7<sup>th</sup> Ed.**

©Silberschatz, Korth and Sudarshan

See [www.db-book.com](http://www.db-book.com) for conditions on re-use



# Outline

- Join Expressions
- Views
- Transactions
- Integrity Constraints
- SQL Data Types and Schemas
- Index Definition in SQL
- Authorization



# Joined Relations

- **Join operations** take two relations and return as a result another relation.
- A join operation is a Cartesian product which requires that tuples in the two relations match (under some condition). It also specifies the attributes that are present in the result of the join
- The join operations are typically used as subquery expressions in the **from** clause
- Three types of joins:
  - Natural join
  - Inner join
  - Outer join



# Natural Join in SQL

- Natural join matches tuples with the same values for **all common attributes**, and retains only one copy of each common column.
- List the ID and names of students along with the course IDs of the courses that they have taken.
  - **select** *student.ID* , *name*, *course\_id*  
**from** *students*, *takes*  
**where** *student.ID* = *takes.ID*;
- Same query in SQL with “natural join” construct
  - **select** *ID*, *name*, *course\_id*  
**from** *student natural join takes*;
- 注意：有些軟體不支援此語法
- 補充Relational algebra 的語法:

$$\Pi_{ID, name, course\_id} (students \bowtie takes)$$



## Natural Join in SQL (Cont.)

- The **from** clause can have multiple relations combined using natural join:

```
select  $A_1, A_2, \dots A_n$   
from  $r_1$  natural join  $r_2$  natural join .. natural join  $r_n$   
where  $P$  ;
```



## Student Relation

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>tot_cred</i>
00128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
19991	Brandt	History	80
23121	Chavez	Finance	110
44553	Peltier	Physics	56
45678	Levy	Physics	46
54321	Williams	Comp. Sci.	54
55739	Sanchez	Music	38
70557	Snow	Physics	0
76543	Brown	Comp. Sci.	58
76653	Aoi	Elec. Eng.	60
98765	Bourikas	Elec. Eng.	98
98988	Tanaka	Biology	120

## Takes Relation

<i>ID</i>	<i>course_id</i>	<i>sec_id</i>	<i>semester</i>	<i>year</i>	<i>grade</i>
00128	CS-101	1	Fall	2017	A
00128	CS-347	1	Fall	2017	A-
12345	CS-101	1	Fall	2017	C
12345	CS-190	2	Spring	2017	A
12345	CS-315	1	Spring	2018	A
12345	CS-347	1	Fall	2017	A
19991	HIS-351	1	Spring	2018	B
23121	FIN-201	1	Spring	2018	C+
44553	PHY-101	1	Fall	2017	B-
45678	CS-101	1	Fall	2017	F
45678	CS-101	1	Spring	2018	B+
45678	CS-319	1	Spring	2018	B
54321	CS-101	1	Fall	2017	A-
54321	CS-190	2	Spring	2017	B+
55739	MU-199	1	Spring	2018	A-
76543	CS-101	1	Fall	2017	A
76543	CS-319	2	Spring	2018	A
76653	EE-181	1	Spring	2017	C
98765	CS-101	1	Fall	2017	C-
98765	CS-315	1	Spring	2018	B
98988	BIO-101	1	Summer	2017	A
98988	BIO-301	1	Summer	2018	<i>null</i>



## *student natural join takes*

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>tot_cred</i>	<i>course_id</i>	<i>sec_id</i>	<i>semester</i>	<i>year</i>	<i>grade</i>
00128	Zhang	Comp. Sci.	102	CS-101	1	Fall	2017	A
00128	Zhang	Comp. Sci.	102	CS-347	1	Fall	2017	A-
12345	Shankar	Comp. Sci.	32	CS-101	1	Fall	2017	C
12345	Shankar	Comp. Sci.	32	CS-190	2	Spring	2017	A
12345	Shankar	Comp. Sci.	32	CS-315	1	Spring	2018	A
12345	Shankar	Comp. Sci.	32	CS-347	1	Fall	2017	A
19991	Brandt	History	80	HIS-351	1	Spring	2018	B
23121	Chavez	Finance	110	FIN-201	1	Spring	2018	C+
44553	Peltier	Physics	56	PHY-101	1	Fall	2017	B-
45678	Levy	Physics	46	CS-101	1	Fall	2017	F
45678	Levy	Physics	46	CS-101	1	Spring	2018	B+
45678	Levy	Physics	46	CS-319	1	Spring	2018	B
54321	Williams	Comp. Sci.	54	CS-101	1	Fall	2017	A-
54321	Williams	Comp. Sci.	54	CS-190	2	Spring	2017	B+
55739	Sanchez	Music	38	MU-199	1	Spring	2018	A-
76543	Brown	Comp. Sci.	58	CS-101	1	Fall	2017	A
76543	Brown	Comp. Sci.	58	CS-319	2	Spring	2018	A
76653	Aoi	Elec. Eng.	60	EE-181	1	Spring	2017	C
98765	Bourikas	Elec. Eng.	98	CS-101	1	Fall	2017	C-
98765	Bourikas	Elec. Eng.	98	CS-315	1	Spring	2018	B
98988	Tanaka	Biology	120	BIO-101	1	Summer	2017	A
98988	Tanaka	Biology	120	BIO-301	1	Summer	2018	<i>null</i>





# Dangerous in Natural Join

- Beware of unrelated attributes with same name which get equated incorrectly
- Example -- List the names of students along with the titles of courses that they have taken

- Correct version

```
select name, title
from student natural join takes, course
where takes.course_id = course.course_id;
```

student ( <u>ID</u> , name, dept_name, tot_cred)
takes ( <u>ID</u> , <u>course_id</u> , <u>sec_id</u> , semester, year, grade)
course ( <u>course_id</u> , title, dept_name, credits)

- 在**from**裡使用逗號“, ”，所以**course**是和前一個**combined relation**做**cartesian product**，而非**natural join**.

- Incorrect version

```
select name, title
from student natural join takes natural join course;
```

- This query omits all (student.name, course.title) pairs where the student takes a course in a department other than the student's own department. <- 錯誤的原因在於多了**student.dept\_name= course.dept\_name**的限制，但**student**和**course**不需要同一系!
- The correct version (above), correctly outputs such pairs.





# Natural Join with Using Clause

- To avoid the danger of equating attributes erroneously, we can use the “**using**” construct that allows us to specify exactly which columns should be equated.

- Query example

```
select name, title  
from (student natural join takes) join course using (course_id)
```

➤ 後面的natural join 改用 join...using(...)

- 注意: 多個表格進行join時, 要注意同名的屬性是否意義相同, 以及是否要求其值相同的才能配對輸出。



# Join Condition

- The **on** condition allows a general predicate over the relations being joined
- This predicate is written like a **where** clause predicate except for the use of the keyword **on**
- Query example

```
select *  
from student join takes on student.ID = takes.ID
```

  - The **on** condition above specifies that a tuple from *student* matches a tuple from *takes* if their *ID* values are equal.
  - “join” 之前省略 “inner” 是OK的 (參考下一頁的說明)
- Equivalent to:

```
select *  
from student , takes  
where student.ID = takes.ID
```
- 注意: on後面不要寫個別屬性值的限制



# Outer Join

- An extension of the join operation that avoids loss of information.
- Computes the inner join and then adds tuples from one relation that does not match tuples in the other relation to the result of the join.
- Uses *null* values.
- Three forms of outer join:
  - left outer join
  - right outer join
  - full outer join
- Note: the join operations we studied earlier that do not preserve nonmatched tuples are called **inner join** operations.
- Inner join的概念類似交集, outer join的概念類似聯集。



# Outer Join Examples

- Relation *course*

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

- Relation *prereq*

<i>course_id</i>	<i>prereq_id</i>
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

- Observe that

*course* information is missing CS-437

*prereq* information is missing CS-315

- $\text{course} \bowtie \text{prereq}$  會沒有CS-315和CS-437的任何資料.



## Example of Left/Right Outer Join

- *course natural left outer join prereq*

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>	<i>prereq_id</i>
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	<i>null</i>

- 注意: “**natural**”會要求兩個relation的共同屬性其值相等.
- *course natural right outer join prereq*

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>	<i>prereq_id</i>
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	<i>null</i>	<i>null</i>	<i>null</i>	CS-101



# Full Outer Join

- *course* **natural full outer join** *prereq*

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>	<i>prereq_id</i>
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	<i>null</i>
CS-347	<i>null</i>	<i>null</i>	<i>null</i>	CS-101

- 注意: 有些軟體不支援full outer join



# Joined Types and Conditions

- **Join operations** take two relations and return as a result another relation.
- These additional operations are typically used as subquery expressions in the **from** clause
- **Join condition** – defines which tuples in the two relations match, and what attributes are present in the result of the join.
- **Join type** – defines how tuples in each relation that do not match any tuple in the other relation (based on the join condition) are treated.

<i>Join types</i>
<b>inner join</b>
<b>left outer join</b>
<b>right outer join</b>
<b>full outer join</b>

<i>Join conditions</i>
<b>natural</b>
<b>on</b> <predicate>
<b>using</b> ( $A_1, A_2, \dots, A_n$ )

- 各軟體的功能不盡相同, 大部分的軟體支援 (inner, left outer, right outer) join + on <predicate>





## Joined Relations – Examples

- **course natural right outer join prereq**

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>	<i>prereq_id</i>
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	<i>null</i>	<i>null</i>	<i>null</i>	CS-101

- **course full outer join prereq using (course\_id)**

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>	<i>prereq_id</i>
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	<i>null</i>
CS-347	<i>null</i>	<i>null</i>	<i>null</i>	CS-101



# Joined Relations – Examples

- *course* **inner join** *prereq* on *course.course\_id* = *prereq.course\_id*

Diagram illustrating an inner join between *course* and *prereq* on the condition *course.course\_id* = *prereq.course\_id*. The result table shows only the rows where the *course\_id* in the *course* table matches the *course\_id* in the *prereq* table.

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>	<i>prereq_id</i>	<i>course_id</i>
BIO-301	Genetics	Biology	4	BIO-101	BIO-301
CS-190	Game Design	Comp. Sci.	4	CS-101	CS-190

- What is the difference between the above, and a natural join?
- *course* **left outer join** *prereq* on *course.course\_id* = *prereq.course\_id*

Diagram illustrating a left outer join between *course* and *prereq* on the condition *course.course\_id* = *prereq.course\_id*. The result table shows all rows from the *course* table, including those that do not have a matching row in the *prereq* table (indicated by *null* values).

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>	<i>prereq_id</i>	<i>course_id</i>
BIO-301	Genetics	Biology	4	BIO-101	BIO-301
CS-190	Game Design	Comp. Sci.	4	CS-101	CS-190
CS-315	Robotics	Comp. Sci.	3	<i>null</i>	<i>null</i>

- 注意: The joined relations are used as subquery expressions in the **from** clause



## 練習

- Find the `course_ids` and `sec_ids` which were offered by an instructor named “Einstein”, using the “inner join” expression.

Answer:

(1) using “natural” expression

(2) Using “on” expression

(3) Using “using” expression



# Views

- In some cases, it is not desirable for all users to see the entire logical model (that is, all the actual relations stored in the database.)
- Consider a person who needs to know an instructor's name and department, but not the salary. This person should see a relation described, in SQL, by

```
select ID, name, dept_name  
from instructor
```

- A **view** provides a mechanism to hide certain data from the view of certain users.
- Any relation that is not part of the conceptual model but is made visible to a user as a “virtual relation” is called a **view**.



# View Definition

- A view is defined using the **create view** statement which has the form

**create view *v* as** < query expression >

where <query expression> is any legal SQL expression. The view name is represented by *v*.

- Once a view is defined, the view name can be used to refer to the virtual relation that the view generates.
- 注意: 此處所定義的view可以被多個查詢句使用, 和with所定義的不同.
- View definition is not the same as creating a new relation by evaluating the query expression
  - Rather, a view definition causes the saving of an expression; the expression is substituted into queries using the view.



# View Definition and Use

- A view of instructors without their salary

```
create view faculty as  
    select ID, name, dept_name  
    from instructor
```

- Find all instructors in the Biology department

```
select name  
from faculty  
where dept_name = 'Biology'
```

- Create a view of department salary totals

```
create view departments_total_salary(dept_name, total_salary) as  
    select dept_name, sum (salary)  
    from instructor  
    group by dept_name;
```



# Views Defined Using Other Views

- create view ***physics\_fall\_2017*** as  
select *course.course\_id, sec\_id, building, room\_number*  
from *course, section*  
where *course.course\_id = section.course\_id*  
and *course.dept\_name = 'Physics'*  
and *section.semester = 'Fall'*  
and *section.year = '2017'*;
- create view ***physics\_fall\_2017\_watson*** as  
select *course\_id, room\_number*  
from ***physics\_fall\_2017***  
where *building= 'Watson'*;





# View Expansion

- Expand the view :

```
create view physics_fall_2017_watson as  
  select course_id, room_number  
  from physics_fall_2017  
  where building= 'Watson'
```

- To:

```
create view physics_fall_2017_watson as  
  select course_id, room_number  
  from (select course.course_id, building, room_number  
        from course, section  
        where course.course_id = section.course_id  
              and course.dept_name = 'Physics'  
              and section.semester = 'Fall'  
              and section.year = '2017')  
  where building= 'Watson';
```



## ⌘ Views Defined Using Other Views

- One view may be used in the expression defining another view
- A view relation  $v_1$  is said to **depend directly** on a view relation  $v_2$  if  $v_2$  is used in the expression defining  $v_1$
- A view relation  $v_1$  is said to **depend on** view relation  $v_2$  if either  $v_1$  depends directly to  $v_2$  or there is a path of dependencies from  $v_1$  to  $v_2$
- A view relation  $v$  is said to be **recursive** if it depends on itself.



## ⌘ View Expansion

- A way to define the meaning of views defined in terms of other views.
- Let view  $v_1$  be defined by an expression  $e_1$  that may itself contain uses of view relations.
- View expansion of an expression repeats the following replacement step:
  - repeat**
    - Find any view relation  $v_i$  in  $e_1$
    - Replace the view relation  $v_i$  by the expression defining  $v_i$
  - until** no more view relations are present in  $e_1$
- As long as the view definitions are not recursive, this loop will terminate



## ⌘ Materialized Views

- Certain database systems allow view relations to be physically stored.
  - Physical copy created when the view is defined.
  - Such views are called **Materialized view**:
- If relations used in the query are updated, the materialized view result becomes out of date
  - Need to **maintain** the view, by updating the view whenever the underlying relations are updated.



# Update of a View

- Add a new tuple to *faculty* view which we defined earlier  
**insert into *faculty***  
**values** ('30765', 'Green', 'Music');
- This insertion must be represented by the insertion into the *instructor* relation
  - Must have a value for salary.
- Two approaches
  - Reject the insert
  - Insert the tuple  
( '30765', 'Green', 'Music', null )  
into the *instructor* relation



# Some Updates Cannot be Translated Uniquely

- **create view *instructor\_info* as**  
     **select** *ID, name, building*  
     **from** *instructor, department*  
     **where** *instructor.dept\_name= department.dept\_name;*
- **insert into *instructor\_info***  
     **values** ('69987', 'White', 'Taylor');
- Issues
  - Which department, if multiple departments in Taylor?
  - What if no department is in Taylor?

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>	<i>dept_name</i>	<i>building</i>	<i>budget</i>
10101	Srinivasan	Comp. Sci.	65000	Biology	Watson	90000
12121	Wu	Finance	90000	Comp. Sci.	Taylor	100000
15151	Mozart	Music	40000	Elec. Eng.	Taylor	85000
22222	Einstein	Physics	95000	Finance	Painter	120000
32343	El Said	History	60000	History	Painter	50000
33456	Gold	Physics	87000	Music	Packard	80000
45565	Katz	Comp. Sci.	75000	Physics	Watson	70000
58583	Califieri	History	62000			
76543	Singh	Finance	80000			
76766	Crick	Biology	72000			
83821	Brandt	Comp. Sci.	92000			
98345	Kim	Elec. Eng.	80000			
69987	White	<i>null</i>	<i>null</i>	<i>null</i>	Taylor	<i>null</i>



## And Some Not at All

- **create view** *history\_instructors* **as**  
    **select** \*  
    **from** *instructor*  
    **where** *dept\_name*= 'History';
- What happens if we insert  
    ('25566', 'Brown', 'Biology', 100000)  
    into *history\_instructors*?





# View Updates in SQL

- Most SQL implementations allow updates only on simple views
  - The **from** clause has only one database relation.
  - The **select** clause contains only attribute names of the relation, and does not have any expressions, aggregates, or **distinct** specification.
  - Any attribute not listed in the **select** clause can be set to null
  - The query does not have a **group** by or **having** clause.



# Transactions

- A **transaction** consists of a sequence of query and/or update statements and is a “unit” of work
- The SQL standard specifies that a transaction begins implicitly when an SQL statement is executed.
- The transaction must end with one of the following statements:
  - **Commit work**. The updates performed by the transaction become permanent in the database.
  - **Rollback work**. All the updates performed by the SQL statements in the transaction are undone.
- Atomic transaction
  - either fully executed or rolled back as if it never occurred
- Isolation from concurrent transactions
- 第五章會有進一步SQL語法和API的用法說明。



# Integrity Constraints

- Integrity constraints guard against accidental damage to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency.
  - A checking account must have a balance greater than \$10,000.00
  - A salary of a bank employee must be at least \$4.00 an hour
  - A customer must have a (non-null) phone number
- 注意: When an integrity constraint is **violated**, the normal procedure is to **reject the action** that caused the violation.
- 大部分資料庫軟體支援的兩種 integrity constraints 類型
  - Constraints on a single relation
  - Constraints on two relations (Referential Integrity constraints)



# Constraints on a Single Relation

- **not null**
- **primary key**
- **unique**
- **check (P)**, where P is a predicate



# Not Null Constraints

- **not null**
  - Declare *name* and *budget* to be **not null**  
*name* **varchar(20) not null**  
*budget* **numeric(12,2) not null**



# Unique Constraints

- **unique** (  $A_1, A_2, \dots, A_m$  )
  - The unique specification states that the attributes  $A_1, A_2, \dots, A_m$  form a candidate key.
  - Candidate keys are permitted to be null (in contrast to primary keys).



# The check clause

- The **check** (P) clause specifies a predicate P that must be satisfied by every tuple in a relation.
- Example: ensure that semester is one of fall, winter, spring or summer

**create table** *section*

```
(course_id varchar (8),  
  sec_id varchar (8),  
  semester varchar (6),  
  year numeric (4,0),  
  building varchar (15),  
  room_number varchar (7),  
  time slot id varchar (4),  
  primary key (course_id, sec_id, semester, year),  
  check (semester in ('Fall', 'Winter', 'Spring', 'Summer')))
```

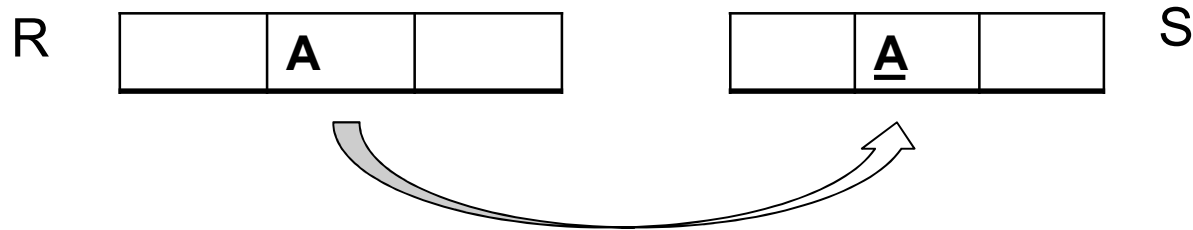
- 有的資料庫軟體允許此功能在資料庫裡宣告，有的則只能利用應用程式，如利用 asp.net 的物件定義。





# Referential Integrity

- Ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation.
  - Example: If “Biology” is a department name appearing in one of the tuples in the *instructor* relation, then there exists a tuple in the *department* relation for “Biology”.
- Let A be a set of attributes. Let R and S be two relations that contain attributes A and where A is the primary key of S. A is said to be a **foreign key** of R if for any values of A appearing in R these values also appear in S.





## Referential Integrity (Cont.)

- Foreign *keys can be* specified as part of the SQL **create table** statement  
**foreign key** (*dept\_name*) **references** *department*
- By default, a foreign key references the primary-key attributes of the referenced table.  
**foreign key** (*dept\_name*) **references** *department* (*dept\_name*)
- SQL allows a list of attributes of the referenced relation to be specified explicitly.
- 此限制對新增、刪除、修改的動作，都可能會影響到。



## 練習

- For the following two tables, suppose *branch\_name* in the *account* relation is a foreign key referencing the *branch* table. When will the referential constraint be violated? (Consider the case of *insert*, *delete*, *update*).

branch	<i>branch_name</i>	<i>branch_city</i>	<i>assets</i>
	Perryridge	Horseneck	1700000
	Downtown	Brooklyn	9000000
	Brighton	Brooklyn	7100000
	Mianus	Horseneck	400000
	Redwood	Palo Alto	2100000

account	<i>branch_name</i>	<i>account_number</i>	<i>balance</i>
	Perryridge	A-102	400
	Perryridge	A-201	900
	Brighton	A-217	750
	Brighton	A-215	750
	Redwood	A-222	700



# Cascading Actions in Referential Integrity

- When a referential-integrity constraint is violated, the normal procedure is to reject the action that caused the violation.
- An alternative, in case of delete or update is to cascade

```
create table course (  
    (...  
    dept_name varchar(20),  
    foreign key (dept_name) references department  
        on delete cascade  
        on update cascade,  
    ...)
```

delete和update可以不一樣

- Instead of cascade we can use :
  - **set null,**
  - **set default**
- 注意: 不同軟體支援的選項可能會有些微的差異



## ✘ Integrity Constraint Violation During Transactions

- Consider:

```
create table person (  
    ID char(10),  
    name char(40),  
    mother char(10),  
    father char(10),  
    primary key ID,  
    foreign key father references person,  
    foreign key mother references person)
```

ID	name	mother	father
2	John	null	null
3	Mary	null	Null
10	Tom	3	2

- How to insert a tuple without causing constraint violation?
  - Insert father and mother of a person before inserting person
  - OR, set father and mother to null initially, update after inserting all persons (not possible if father and mother attributes declared to be **not null**)
  - OR defer constraint checking:  
**set constraints** constraint-list **deferred (commit時才check)**



## ⌘ Complex Check Conditions

- The predicate in the check clause can be an arbitrary predicate that can include a subquery.

**check** (*time\_slot\_id* in (**select** *time\_slot\_id* **from** *time\_slot*))

The check condition states that the *time\_slot\_id* in each tuple in the *section* relation is actually the identifier of a time slot in the *time\_slot* relation.

- The condition has to be checked not only when a tuple is inserted or modified in *section* , but also when the relation *time\_slot* changes



# Built-in Data Types in SQL

- **date:** Dates, containing a (4 digit) year, month and date
  - Example: **date** '2005-7-27'
- **time:** Time of day, in hours, minutes and seconds.
  - Example: **time** '09:00:30'      **time** '09:00:30.75'
- **timestamp:** date plus time of day
  - Example: **timestamp** '2005-7-27 09:00:30.75'
- **interval:** period of time
  - Example: **interval** '1' day
  - Subtracting a date/time/timestamp value from another gives an interval value
  - Interval values can be added to date/time/timestamp values
- 各資料庫軟體的做法不同,一般利用程式語言提供的函數或類別實做下述功能:
  - functions to convert a character string to the type above.
  - functions to get the current time.



# Large-Object Types

- Large objects (photos, videos, CAD files, etc.) are stored as a *large object*:
  - **blob**: binary large object -- object is a large collection of uninterpreted binary data (whose interpretation is left to an application outside of the database system)
  - **clob**: character large object -- object is a large collection of character data
- When a query returns a large object, a pointer is returned rather than the large object itself.
- 補充：In MS SQL，提供binary, image, text 等型態





# User-Defined Types

- **create type** construct in SQL creates user-defined type

**create type** *Dollars* as numeric (12,2) final

- Example:

```
create table department  
  (dept_name varchar (20),  
   building varchar (15),  
   budget Dollars);
```



# Domains

- **create domain** construct in SQL-92 creates user-defined domain types

**create domain** *person\_name* **char(20) not null**

- Types and domains are similar. Domains can have constraints, such as **not null**, specified on them.
- Example:

```
create domain degree_level varchar(10)  
constraint degree_level_test  
check (value in ('Bachelors', 'Masters', 'Doctorate'));
```

- 補充說明: The clause **constraint** is optional; it can give a name to the constraint.
- 各軟體支援的type/domain功能和語法差異很大.

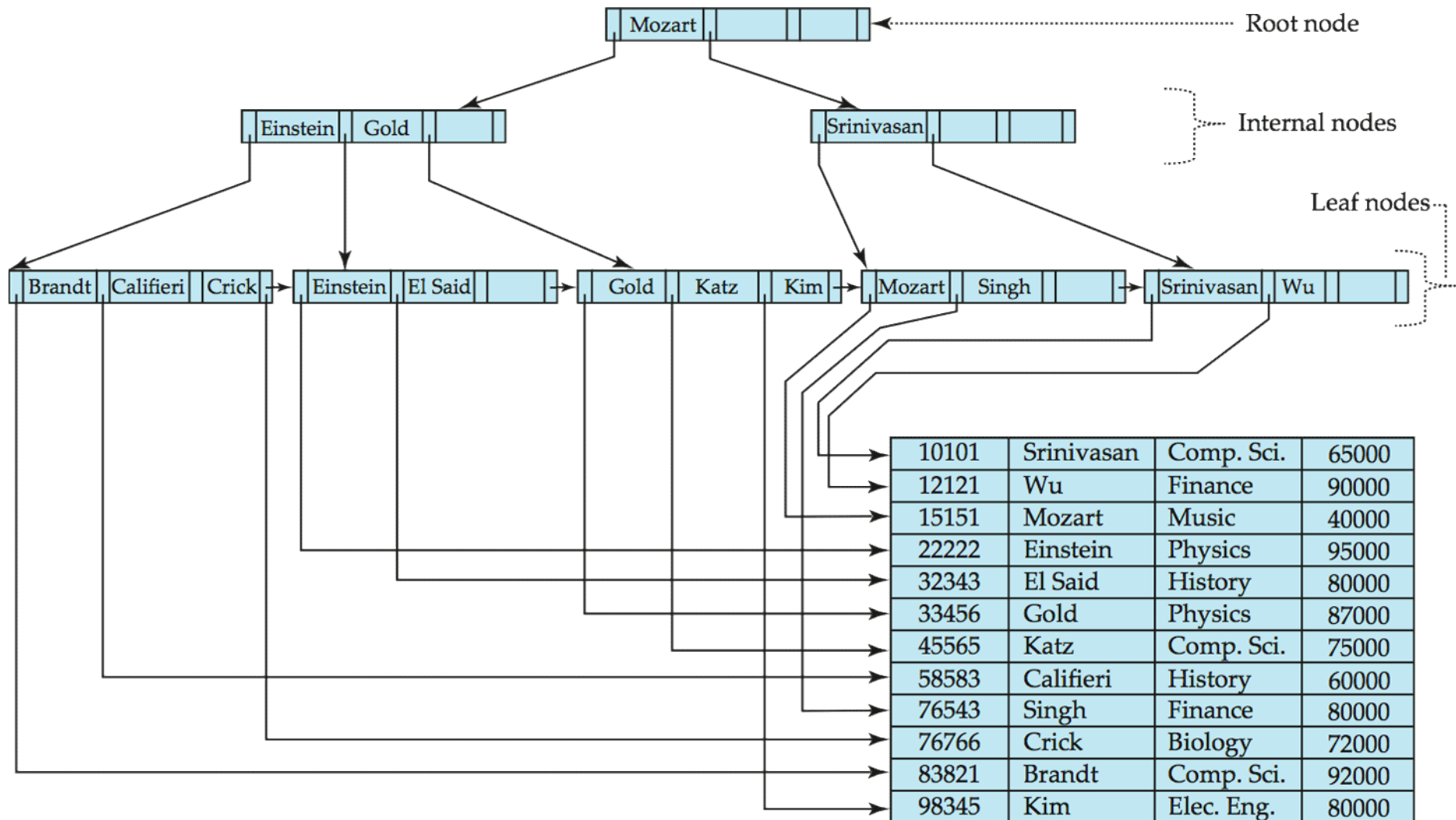


# Index Creation

- Many queries reference only a small proportion of the records in a table.
- It is inefficient for the system to read every record to find a record with particular value
- An **index** on an attribute of a relation is a data structure that allows the database system to find those tuples in the relation that have a specified value for that attribute efficiently, without scanning through all the tuples of the relation.
- We create an index with the **create index** command  
**create index** <name> **on** <relation-name> (attribute);



# 以屬性name建立Index的範例圖 (含Data File)





# Index Creation Example

- **create table** *student*  
(*ID* **varchar** (5),  
*name* **varchar** (20) **not null**,  
*dept\_name* **varchar** (20),  
*tot\_cred* **numeric** (3,0) **default** 0,  
**primary key** (*ID*))
- **create index** *studentID\_index* **on** *student*(*ID*)

- The query:

```
select *  
from student  
where ID = '12345'
```

can be executed by using the index to find the required record, without looking at all records of *student*



# Authorization

- We may assign a user several forms of authorizations on parts of the database.
  - **Read** - allows reading, but not modification of data.
  - **Insert** - allows insertion of new data, but not modification of existing data.
  - **Update** - allows modification, but not deletion of data.
  - **Delete** - allows deletion of data.
- Each of these types of authorizations is called a **privilege**. We may authorize the user all, none, or a combination of these types of privileges on specified parts of a database, such as a relation or a view.



## Authorization (Cont.)

- Forms of authorization to modify the database schema
  - **Index** - allows creation and deletion of indices.
  - **Resources** - allows creation of new relations.
  - **Alteration** - allows addition or deletion of attributes in a relation.
  - **Drop** - allows deletion of relations.



# Privileges in SQL

- **select**: allows read access to relation, or the ability to query using the view
  - **insert**: the ability to insert tuples
  - **update**: the ability to update using the SQL update statement
  - **delete**: the ability to delete tuples.
  - **all privileges**: used as a short form for all the allowable privileges
- (PS. 還有很多其他privileges, 各家軟體不同)





# Authorization Specification in SQL

- The **grant** statement is used to confer authorization  
**grant** <privilege list> **on** <relation or view > **to** <user list>
- <user list> is:
  - a user-id (資料庫的帳號)
  - **public**, which allows all valid users the privilege granted
  - A role (more on this later)
- Example: (Amit等是user id)
  - **grant select on department to** Amit, Satoshi
- Granting a privilege on a view does not imply granting any privileges on the underlying relations.
- The grantor of the privilege must already hold the privilege on the specified item (or be the database administrator).



# Revoking Authorization in SQL

- The **revoke** statement is used to revoke authorization.  
**revoke** <privilege list> **on** <relation or view> **from** <user list>
- Example:  
**revoke select on student from  $U_1, U_2, U_3$**
- <privilege-list> may be **all** to revoke all privileges the revokee may hold.
- If <revokee-list> includes **public**, all users lose the privilege except those granted it explicitly.
- If the same privilege was granted twice to the same user by different grantees, the user may retain the privilege after the revocation.
- All privileges that depend on the privilege being revoked are also revoked.



# Roles

- A **role** is a way to distinguish among various users as far as what these users can access/update in the database.
- To create a role we use:
  - create role** <name>
- Example:
  - **create role** instructor
- Once a role is created we can assign “users” to the role using:
  - **grant** <role> **to** <users>
- Example:
  - **create role** instructor;
  - **grant** *instructor* **to** Amit;



# Roles Example

- Privileges can be granted to roles:
  - **grant select on *takes* to *instructor*;**
- Roles can be granted to users, as well as to other roles
  - **create role *teaching\_assistant***
  - **grant *teaching\_assistant* to *instructor*;**
    - *Instructor* inherits all privileges of *teaching\_assistant*
- Chain of roles
  - **create role *dean*;**
  - **grant *instructor* to *dean*;**
  - **grant *dean* to Satoshi;**



# Authorization on Views

- **create view** *geo\_instructor* as  
(**select** \*  
**from** *instructor*  
**where** *dept\_name* = 'Geology');
- **grant select on** *geo\_instructor* **to** *geo\_staff*
- Suppose that a *geo\_staff* member issues
  - **select** \*  
**from** *geo\_instructor*;
- This staff is authorized to see the result of the above query.