

More SQL

Extended Relational Algebra
Outerjoins, Grouping/Aggregation
Insert/Delete/Update

The Extended Algebra

δ = eliminate duplicates from bags.

τ = sort tuples.

γ = grouping and aggregation.

Outerjoin : avoids “dangling tuples” = tuples that do not join with anything.

Duplicate Elimination

- ◆ $R1 := \delta(R2).$
- ◆ R1 consists of one copy of each tuple that appears in R2 one or more times.

Example: Duplicate Elimination

$R =$ (

A	B
1	2
3	4
1	2

)

$\delta(R) =$

A	B
1	2
3	4

Sorting

- ◆ $R1 := \tau_L(R2)$.
 - ◆ L is a list of some of the attributes of $R2$.
- ◆ $R1$ is the list of tuples of $R2$ sorted first on the value of the first attribute on L , then on the second attribute of L , and so on.
 - ◆ Break ties arbitrarily.
- ◆ τ is the only operator whose result is neither a set nor a bag.

Example: Sorting

$R =$ (

A	B
1	2
3	4
5	2

)

$$\tau_B(R) = [(5,2), (1,2), (3,4)]$$

Aggregation Operators

- ◆ Aggregation operators are not operators of relational algebra.
- ◆ Rather, they apply to entire columns of a table and produce a single result.
- ◆ The most important examples: SUM, AVG, COUNT, MIN, and MAX.

Example: Aggregation

R = (

A	B
1	3
3	4
3	2

)

SUM(A) = 7

COUNT(A) = 3

MAX(B) = 4

AVG(B) = 3

Grouping Operator

- ◆ $R1 := \gamma_L (R2)$. L is a list of elements that are either:
 1. Individual (*grouping*) attributes.
 2. $AGG(A)$, where AGG is one of the aggregation operators and A is an attribute.
 - An arrow and a new attribute name renames the component.

Applying $\gamma_L(R)$

- ◆ Group R according to all the grouping attributes on list L .
 - ◆ That is: form one group for each distinct list of values for those attributes in R .
- ◆ Within each group, compute $AGG(A)$ for each aggregation on list L .
- ◆ Result has one tuple for each group:
 1. The grouping attributes and
 2. Their group's aggregations.

Example: Grouping/Aggregation

$R =$ (

A	B	C
1	2	3
4	5	6
1	2	5

Then, average C
within groups:

A	B	X
1	2	4
4	5	6

$\gamma_{A,B,AVG(C) \rightarrow X}(R) = ??$

First, group R by A and B :

A	B	C
1	2	3
1	2	5
4	5	6

Outerjoin

- ◆ Suppose we join $R \bowtie_c S$.
- ◆ A tuple of R that has no tuple of S with which it joins is said to be *dangling*.
 - ◆ Similarly for a tuple of S .
- ◆ Outerjoin preserves dangling tuples by padding them NULL.

Example: Outerjoin

R = (

A	B
1	2
4	5

S = (

B	C
2	3
6	7

(1,2) joins with (2,3), but the other two tuples are dangling.

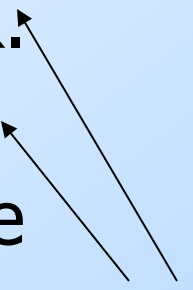
R OUTERJOIN S =

A	B	C
1	2	3
4	5	NULL
NULL	6	7

Now --- Back to SQL

Each Operation Has a SQL
Equivalent

Outerjoins

- ◆ R OUTER JOIN S is the core of an outerjoin expression. It is modified by:
 1. Optional NATURAL in front of OUTER.
 2. Optional ON <condition> after JOIN.
 3. Optional LEFT, RIGHT, or FULL before OUTER.
 - ◆ LEFT = pad dangling tuples of R only.
 - ◆ RIGHT = pad dangling tuples of S only.
 - ◆ FULL = pad both; this choice is the default.
- Only one of these
- 

Aggregations

- ◆ SUM, AVG, COUNT, MIN, and MAX can be applied to a column in a SELECT clause to produce that aggregation on the column.
- ◆ Also, COUNT(*) counts the number of tuples.

Aggregate function

<i>Aggregate function</i>	<i>Description</i>
AVG	Average of values in a numeric expression
COUNT	Number of values in an expression
COUNT (*)	Number of selected rows
MAX	Highest value in the expression
MIN	Lowest value in the expression
SUM	Total values in a numeric expression
STDEV	Statistical deviation of all values
STDEVP	Statistical deviation for the population
VAR	Statistical variance of all values
VARP	Statistical variance of all values for the population

Example: Aggregation

- ◆ From **Sells(bar, beer, price)**, find the average price of Bud:

```
SELECT AVG (price)
FROM Sells
WHERE beer = 'Bud' ;
```

COUNT

- ◆ 统计表行数
 - `SELECT COUNT (*) FROM employees`
- ◆ 统计不含空值的行数
 - `SELECT COUNT(reportsto) FROM employees`

Eliminating Duplicates in an Aggregation

- ◆ Use DISTINCT inside an aggregation.
- ◆ **Example:** find the number of *different* prices charged for Bud:

```
SELECT COUNT(DISTINCT price)
FROM Sells
WHERE beer = 'Bud';
```

NULL's Ignored in Aggregation

- ◆ NULL never contributes to a sum, average, or count, and can never be the minimum or maximum of a column.
- ◆ But if there are no non-NULL values in a column, then the result of the aggregation is NULL.
 - ◆ **Exception:** COUNT of an empty set is 0.

Example: Effect of NULL's

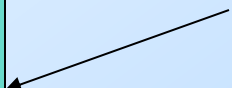
```
SELECT count(*)  
FROM Sells  
WHERE beer = 'Bud';
```

The number of bars
that sell Bud.



```
SELECT count(price)  
FROM Sells  
WHERE beer = 'Bud';
```

The number of bars
that sell Bud at a
known price.



Grouping

- ◆ We may follow a SELECT-FROM-WHERE expression by GROUP BY and a list of attributes.
- ◆ The relation that results from the SELECT-FROM-WHERE is grouped according to the values of all those attributes, and any aggregation is applied only within each group.

Example: Grouping

- ◆ From `Sells(bar, beer, price)`, find the average price for each beer:

```
SELECT beer, AVG(price)
FROM Sells
GROUP BY beer;
```

beer	AVG(price)
Bud	2.33
...	...

Example: Grouping

- ◆ From `Sells(bar, beer, price)` and `Frequents(drinker, bar)`, find for each drinker the average price of Bud at the bars they frequent:

```
SELECT drinker, AVG(price)
```

```
FROM Frequents, Sells  
WHERE beer = 'Bud' AND  
      Frequents.bar = Sells.bar
```

```
GROUP BY drinker;
```

Compute all drinker-bar-price triples for Bud.

Then group them by drinker.

Using the GROUP BY Clause

```
SELECT productid, orderid, quantity
FROM orderhist
```

<i>productid</i>	<i>orderid</i>	<i>quantity</i>
1	1	5
1	1	10
2	1	10
2	2	25
3	1	15
3	2	30

```
SELECT productid
, SUM(quantity) AS total_quantity
FROM orderhist
GROUP BY productid
```

子句

用聚集函数

Only rows that
satisfy the WHERE
clause are grouped

<i>productid</i>	<i>total_quantity</i>
1	15
2	35
3	45

<i>productid</i>	<i>total_quantity</i>
2	35

```
SELECT productid
, SUM(quantity) AS total_quantity
FROM orderhist
WHERE productid = 2
GROUP BY productid
```

Restriction on SELECT Lists With Aggregation

- ◆ If any aggregation is used, then each element of the SELECT list must be either:
 1. Aggregated, or
 2. An attribute on the GROUP BY list.

Illegal Query Example

- ◆ You might think you could find the bar that sells Bud the cheapest by:

```
SELECT bar, MIN(price)  
FROM Sells  
WHERE beer = 'Bud';
```

- ◆ But this query is illegal in SQL.

HAVING Clauses

- ◆ HAVING <condition> may follow a GROUP BY clause.
- ◆ If so, the condition applies to each group, and groups not satisfying the condition are eliminated.

Using the GROUP BY Clause with the HAVING Clause

```
SELECT productid, orderid, quantity
FROM orderhist
```

```
SELECT productid, SUM(quantity)
      AS total_quantity
FROM orderhist
GROUP BY productid
HAVING SUM(quantity) >= 30
```

◆ 使用 GROUP BY...

productid	orderid	quantity
1	1	5
1	1	10
2	1	10
2	2	25
3	1	15
3	2	30

WHERE子句

用于FROM中

用于组。

不能使用聚集函数；而

HAVING短语中可以使用聚集函数。

productid	total_quantity
2	35
3	45


Example: HAVING

- ◆ From `Sells(bar, beer, price)` and `Beers(name, manf)`, find the average price of those beers that are either served in at least three bars or are manufactured by Pete's.

Solution

```
SELECT beer, AVG(price)
FROM Sells
GROUP BY beer
```


Beer groups with at least 3 non-NULL bars and also beer groups where the manufacturer is Pete's.



```
HAVING COUNT(bar) >= 3 OR
```

```
beer IN (SELECT name
FROM Beers
WHERE manf = 'Pete"s');
```

Beers manufactured by Pete's.



Requirements on HAVING Conditions

- ◆ Anything goes in a subquery.
- ◆ Outside subqueries, they may refer to attributes only if they are either:
 1. A grouping attribute, or
 2. Aggregated(same condition as for SELECT clauses with aggregation).

Database Modifications

- ◆ A *modification* command does not return a result (as a query does), but changes the database in some way.
- ◆ Three kinds of modifications:
 1. *Insert* a tuple or tuples.
 2. *Delete* a tuple or tuples.
 3. *Update* the value(s) of an existing tuple or tuples.

Insertion

- ◆ To insert a single tuple:

```
INSERT INTO <relation>  
VALUES ( <list of values> );
```

- ◆ **Example:** add to **Likes(drinker, beer)**
the fact that Sally likes Bud.

```
INSERT INTO Likes  
VALUES ( 'Sally' , 'Bud' );
```

Specifying Attributes in INSERT

- ◆ We may add to the relation name a list of attributes.
- ◆ Two reasons to do so:
 1. We forget the standard order of attributes for the relation.
 2. We don't have values for all attributes, and we want the system to fill in missing components with NULL or a default value.

Example: Specifying Attributes

- ◆ Another way to add the fact that Sally likes Bud to `Likes(drinker, beer)`:

```
INSERT INTO Likes (beer, drinker)
VALUES ('Bud', 'Sally');
```

Adding Default Values

- ◆ In a CREATE TABLE statement, we can follow an attribute by DEFAULT and a value.
- ◆ When an inserted tuple has no value for that attribute, the default will be used.

Example: Default Values

```
CREATE TABLE Drinkers (  
    name CHAR(30) PRIMARY KEY,  
    addr CHAR(50)  
        DEFAULT '123 Sesame St.',  
    phone CHAR(16)  
);
```

GAUSS数据类型-数值类型-整数

名称	描述	存储空间	范围
TINYINT	微整数，别名为INT1。	1字节	0 ~ 255
SMALLINT	小范围整数，别名为INT2。	2字节	-32,768 ~ +32,767
INTEGER	常用的整数，别名为INT4。	4字节	-2,147,483,648 ~ +2,147,483,647
BINARY_INTEGER	常用的整数INTEGER的别名。	4字节	-2,147,483,648 ~ +2,147,483,647
BIGINT	大范围的整数，别名为INT8。	8字节	-9,223,372,036,854,775,808 ~ +9,223,372,036,854,775,807
int16	十六字节的大范围证书，目前不支持用户用于建表等使用。	16字节	-170,141,183,460,469,231,731,687,303,715,884,105,728 ~ +170,141,183,460,469,231,731,687,303,715,884,105,727

GAUSS数据类型-数值类型-任意精度

名称	描述	存储空间	范围
NUMERIC[(p[,s])], DECIMAL[(p[,s])]	精度p取值范围为[1,1000]，标度s取值范围为[0,p]。 说明：p为总位数，s为小数位数。	用户声明精度。每四位（十进制位）占用两个字节，然后在整个数据上加上八个字节的额外开销。	未指定精度的情况下，小数点前最大 131,072 位，小数点后最大 16,383 位。
NUMBER[(p[,s])]	NUMERIC类型的别名，为兼容Oracle数据类型。	用户声明精度。每四位（十进制位）占用两个字节，然后在整个数据上加上八个字节的额外开销。	未指定精度的情况下，小数点前最大 131,072 位，小数点后最大 16,383 位。

GAUSS数据类型-数值类型-序列整形

名称	描述	存储空间	范围
SMALLSERIAL	二字节序列整形。	2字节	-32,768 ~ +32,767
SERIAL	四字节序列整形。	4字节	-2,147,483,648 ~ +2,147,483,647
BIGSERIAL	八字节序列整形。	8字节	-9,223,372,036,854,775,808 ~ +9,223,372,036,854,775,807

GAUSS数据类型-数值类型-浮点类型

名称	描述	存储空间	范围
REAL, FLOAT4	单精度浮点数，不精准。	4字节	-3.402E+38~3.402E+38 6位十进制数字精度。
DOUBLE PRECISION, FLOAT8	双精度浮点数，不精准。	8字节	-1.79E+308~1.79E+308, 15位十进制数字精度。
FLOAT[(p)]	浮点数，不精准。精度p取值范围为[1,53]。 说明：p为精度，表示二进制总位数。	4字节或8字节	根据精度p不同选择REAL或DOUBLE PRECISION作为内部表示。如不指定精度，内部用DOUBLE PRECISION表示。
BINARY_DOUBLE	是DOUBLE PRECISION的别名，为兼容Oracle类型。	8字节	-1.79E+308~1.79E+308，15位十进制数字精度。
DEC[(p[,s])]	精度p取值范围为[1,1000]，标度s取值范围为[0,p]。 说明：p为总位数，s为小数位位数。	用户声明精度。每四位（十进制位）占用两个字节，然后在整个数据上加上八个字节的额外开销。	未指定精度的情况下，小数点前最大131,072位，小数点后最大16,383位。
INTEGER[(p[,s])]	精度p取值范围为[1,1000]，标度s取值范围为[0,p]。	用户声明精度。每四位（十进制位）占用两个字节，然后在整个数据上加上八个字节的额外开销。	-

GAUSS数据类型-货币类型/逻辑类型

名称	存储容量	描述	范围
money	8 字节	货币金额	-92233720368547758.08 到 +92233720368547758.07

名称	描述	存储空间	取值
BOOLEAN	布尔类型	1字节。	•true: 真 •false: 假 •null: 未知 (unknown)

GAUSS数据类型-字符类型

名称	描述	存储空间
CHAR(n) CHARACTER(n) NCHAR(n)	定长字符串，不足补空格。 n 是指字节长度，如不带精度 n ，默认精度为 1 。	最大为 10MB 。
VARCHAR(n) CHARACTER VARYING(n)	变长字符串。 n 是指字节长度。	最大为 10MB 。
VARCHAR2(n)	变长字符串。是 VARCHAR(n) 类型的别名。 n 是指字节长度。	最大为 10MB 。
NVARCHAR2(n)	变长字符串。 n 是指字符长度。	最大为 10MB 。
TEXT	变长字符串。	最大为 1GB-1 ，但还需要考虑到列描述头信息的大小，以及列所在元组的大小限制（也小于 1GB-1 ），因此 TEXT 类型最大大小可能小于 1GB-1 。
CLOB	文本大对象。是 TEXT 类型的别名。	最大为 1GB-1 ，但还需要考虑到列描述头信息的大小，以及列所在元组的大小限制（也小于 1GB-1 ），因此 CLOB 类型最大大小可能小于 1GB-1 。

GAUSS数据类型-二进制类型

名称	描述	存储空间
BLOB	二进制大对象。 说明：列存不支持BLOB类型。	最大为1GB-8203字节（即1073733621字节）。
RAW	变长的十六进制类型。 说明：列存不支持RAW类型。	4字节加上实际的十六进制字符串。最大为1GB-8203字节（即1073733621字节）。
BYTEA	变长的二进制字符串。	4字节加上实际的二进制字符串。最大为1GB-8203字节（即1073733621字节）。
BYTEAWITHOUTORDERWITHEQUALCOL	变长的二进制字符串（密态特性新增的类型，如果加密列的加密类型指定为确定性加密，则该列的实际类型为BYTEAWITHOUTORDERWITHEQUALCOL），元命令打印加密表将显示原始数据类型。	4字节加上实际的二进制字符串。最大为1GB减去53字节（即1073741771字节）。
BYTEAWITHOUTORDERCOL	变长的二进制字符串（密态特性新增的类型，如果加密列的加密类型指定为随机加密，则该列的实际类型为BYTEAWITHOUTORDERCOL），元命令打印加密表将显示原始数据类型。	4字节加上实际的二进制字符串。最大为1GB减去53字节（即1073741771字节）。
_BYTEAWITHOUTORDERWITHEQUALCOL	变长的二进制字符串，密态特性新增的类型。	4字节加上实际的二进制字符串。最大为1GB减去53字节（即1073741771字节）。
_BYTEAWITHOUTORDERCOL	变长的二进制字符串，密态特性新增的类型。	4字节加上实际的二进制字符串。最大为1GB减去53字节（即1073741771字节）。

GAUSS数据类型-日期时间类型

名称	描述	存储空间
DATE	日期和时间。日期从公元前4713年到公元294276年	4字节（实际存储空间大小为8字节）
TIME [(p)] [WITHOUT TIME ZONE]	只用于一日内时间。 p表示小数点后的精度，取值范围为0~6。	8字节
TIME [(p)] [WITH TIME ZONE]	只用于一日内时间，带时区。 p表示小数点后的精度，取值范围为0~6。	12字节
TIMESTAMP[(p)] [WITHOUT TIME ZONE]	日期和时间。 p表示小数点后的精度，取值范围为0~6。	8字节
TIMESTAMP[(p)] [WITH TIME ZONE]	日期和时间，带时区。TIMESTAMP的别名为TIMESTAMPTZ。 p表示小数点后的精度，取值范围为0~6。	8字节
SMALLDATETIME	日期和时间，不带时区。 精确到分钟，秒位大于等于30秒进一位。	8字节
INTERVAL DAY (l) TO SECOND (p)	时间间隔，X天X小时X分X秒。 •l: 天数的精度，取值范围为0~6。兼容性考虑，目前未实现具体功能。 •p: 秒数的精度，取值范围为0~6。小数末尾的零不显示。	16字节
INTERVAL [FIELDS] [(p)]	时间间隔。 •fields: 可以是YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, DAY TO HOUR, DAY TO MINUTE, DAY TO SECOND, HOUR TO MINUTE, HOUR TO SECOND, MINUTE TO SECOND。 •p: 秒数的精度，取值范围为0~6，且fields为SECOND, DAY TO SECOND, HOUR TO SECOND或MINUTE TO SECOND时，参数p才有效。小数末尾的零不显示。	12字节
reltime	相对时间间隔。格式为： X years X mons X days XX:XX:XX。 采用儒略历计时，规定一年为365.25天，一个月为30天，计算输入值对应的相对时间间隔，输出采用PGRES格式。	4字节
abstime	日期和时间。格式为： YYYY-MM-DD hh:mm:ss+timezone 取值范围为1901-12-13 20:45:53 GMT~2038-01-18 23:59:59 GMT，精度为秒。	4字节

Example: Default Values

```
INSERT INTO Drinkers (name)
VALUES ('Sally');
```

Resulting tuple:

name	address	phone
Sally	123 Sesame St	NULL

Inserting Many Tuples

- ◆ We may insert the entire result of a query into a relation, using the form:

```
INSERT INTO <relation>  
( <subquery> );
```

Example: Insert a Subquery

- ◆ Using `Frequents(drinker, bar)`, enter into the new relation `PotBuddies(name)` all of Sally's "potential buddies," i.e., those drinkers who frequent at least one bar that Sally also frequents.

Solution

The other
drinker

Pairs of Drinker
tuples where the
first is for Sally,
the second is for
someone else,
and the bars are
the same.

INSERT INTO PotBuddies

(SELECT d2.drinker

FROM Frequents d1, Frequents d2
WHERE d1.drinker = 'Sally' AND
d2.drinker <> 'Sally' AND
d1.bar = d2.bar

);

Creating a Table Using the SELECT INTO Statement

- ◆ Use to Create a Table and Insert Rows into the Table in a Single Operation
- ◆ Create a Local or Global Temporary Table
- ◆ Set the select into/bulkcopy Database Option ON in Order to Create a Permanent Table
- ◆ Create Column Alias or Specify Column Names in the Select List for New Table

```
SELECT productname AS products
       ,unitprice AS price
       ,(unitprice * 1.1) AS tax
INTO #pricetable
FROM products
```

Deletion

- ◆ To delete tuples satisfying a condition from some relation:

```
DELETE FROM <relation>  
WHERE <condition>;
```

Example: Deletion

- ◆ Delete from **Likes(drinker, beer)** the fact that Sally likes Bud:

```
DELETE FROM Likes  
WHERE drinker = 'Sally' AND  
      beer = 'Bud';
```

Example: Delete all Tuples

- ◆ Make the relation Likes empty:

```
DELETE FROM Likes;
```

- ◆ Note no WHERE clause needed.

Example: Delete Some Tuples

- ◆ Delete from **Beers(name, manf)** all beers for which there is another beer by the same manufacturer.

```
DELETE FROM Beers b  
WHERE EXISTS (
```

```
SELECT name FROM Beers  
WHERE manf = b.manf AND  
name <> b.name);
```

Beers with the same manufacturer and a different name from the name of the beer represented by tuple b.

Semantics of Deletion --- (1)

- ◆ Suppose Anheuser-Busch makes only Bud and Bud Lite.
- ◆ Suppose we come to the tuple b for Bud first.
- ◆ The subquery is nonempty, because of the Bud Lite tuple, so we delete Bud.
- ◆ Now, when b is the tuple for Bud Lite, do we delete that tuple too?

Semantics of Deletion --- (2)

- ◆ **Answer:** we *do* delete Bud Lite as well.
- ◆ The reason is that deletion proceeds in two stages:
 1. Mark all tuples for which the WHERE condition is satisfied.
 2. Delete the marked tuples.

Updates

- ◆ To change certain attributes in certain tuples of a relation:

UPDATE <relation>

SET <list of attribute assignments>

WHERE <condition on tuples>;

Example: Update

- ◆ Change drinker Fred's phone number to 555-1212:

```
UPDATE Drinkers  
SET phone = '555-1212'  
WHERE name = 'Fred';
```

Example: Update Several Tuples

- ◆ Make \$4 the maximum price for beer:

```
UPDATE Sells  
SET price = 4.00  
WHERE price > 4.00;
```

openGauss备份概述 (1)

- ◆ 数据备份是保护数据安全的重要手段之一，为了更好的保护数据安全，openGauss数据库支持两种备份恢复类型、多种备份恢复方案，备份和恢复过程中提供数据的可靠性保障机制。

openGauss备份概述 (2)

◆备份与恢复类型可分为逻辑备份与恢复、物理备份与恢复。

- ◆ 逻辑备份与恢复：通过逻辑导出对数据进行备份，逻辑备份只能基于备份时刻进行数据转储，所以恢复时也只能恢复到备份时保存的数据。对于故障点和备份点之间的数据，逻辑备份无能为力，逻辑备份适合备份那些很少变化的数据，当这些数据因误操作被损坏时，可以通过逻辑备份进行快速恢复。如果通过逻辑备份进行全库恢复，通常需要重建数据库，导入备份数据来完成，对于可用性要求很高的数据库，这种恢复时间太长，通常不被采用。由于逻辑备份具有平台无关性，所以更为常见的是，逻辑备份被作为一个数据迁移及移动的主要手段。

openGauss备份概述 (3)

◆备份与恢复类型可分为逻辑备份与恢复、物理备份与恢复。

- ◆物理备份与恢复：通过物理文件拷贝的方式对数据库进行备份，以磁盘块为基本单位将数据从主机复制到备机。通过备份的数据文件及归档日志等文件，数据库可以进行完全恢复。物理备份速度快，一般被用作对数据进行备份和恢复，用于全量备份的场景。通过合理规划，可以低成本进行备份与恢复。

openGauss备份概述 (4)

◆ 以下为openGauss支持的两类数据备份恢复方案，备份方案也决定了当异常发生时该如何恢复。

备份类型	应用场景	支持介质	优缺点
逻辑备份与恢复	适合于数据量小的场景。 目前用于表备份恢复，可以备份恢复单表和多表。	<ul style="list-style-type: none">• 磁盘• NBU• OBS• EISOO	可按用户需要进行指定对象的备份和恢复，灵活度高。 当数据量大时，备份效率低。
物理备份与恢复	适用于数据量大的场景，主要用于全量数据备份恢复，也可对整个数据库中的WAL归档日志和运行日志进行备份恢复。 目前用于集群备份恢复，支持集群的全量备份恢复和增量备份恢复。		数据量大时，备份效率高。

openGauss备份概述 (5)

◆当需要进行备份恢复操作时，主要从以下四个方面考虑数据备份方案。

- ◆ 备份对业务的影响在可接受范围。
- ◆ 数据库恢复效率。
 - 为尽量减小数据库故障的影响，要使恢复时间减到最少，从而使恢复的效率达到最高。
- ◆ 数据可恢复程度。
 - 当数据库失效后，要尽量减少数据损失。
- ◆ 数据库恢复成本。

openGauss备份概述 (6)

◆在现网选择备份策略时参考的因素比较多，如备份对象、数据大小、网络配置等，下表列出了可用的备份策略和每个备份策略的适用场景。

备份策略	关键性能因素	典型数据量	性能规格
集群备份	<ul style="list-style-type: none">数据大小网络配置（NBU/EISOO）	数据：PB级 对象：约100万个	备份： <ul style="list-style-type: none">每个主机80 Mbit/s（NBU/EISOO+磁盘）约90%磁盘I/O速率（SSD/HDD）
表备份	<ul style="list-style-type: none">数据库节点数表所在模式网络配置（NBU）	数据：10 TB级	备份：基于查询性能速度+I/O速度。 说明 <ul style="list-style-type: none">多表备份时，备份耗时计算方式：总时间 = 表数量 × 起步时间 + 数据总量 / 数据备份速度。表越小，备份性能更低。

openGauss备份与恢复

- ◆ Ass3、Ass4均涉及openGauss备份与恢复，由于学时原因，课堂不讲授具体语句，请同学们自学。